

# Crazy Eddies GUI System Reference Manual

## 0.6.0

Generated by Doxygen 1.5.4

Sun Mar 16 14:49:30 2008



# Contents

<b>1</b>	<b>Crazy Eddies GUI System Namespace Index</b>	<b>1</b>
1.1	Crazy Eddies GUI System Namespace List . . . . .	1
<b>2</b>	<b>Crazy Eddies GUI System Hierarchical Index</b>	<b>3</b>
2.1	Crazy Eddies GUI System Class Hierarchy . . . . .	3
<b>3</b>	<b>Crazy Eddies GUI System Class Index</b>	<b>11</b>
3.1	Crazy Eddies GUI System Class List . . . . .	11
<b>4</b>	<b>Crazy Eddies GUI System Page Index</b>	<b>23</b>
4.1	Crazy Eddies GUI System Related Pages . . . . .	23
<b>5</b>	<b>Crazy Eddies GUI System Namespace Documentation</b>	<b>25</b>
5.1	CEGUI Namespace Reference . . . . .	25
5.2	CEGUI::CheckboxProperties Namespace Reference . . . . .	55
5.3	CEGUI::ComboboxProperties Namespace Reference . . . . .	56
5.4	CEGUI::EditboxProperties Namespace Reference . . . . .	57
5.5	CEGUI::FontProperties Namespace Reference . . . . .	58
5.6	CEGUI::FrameWindowProperties Namespace Reference . . . . .	59
5.7	CEGUI::ItemEntryProperties Namespace Reference . . . . .	60
5.8	CEGUI::ItemListBaseProperties Namespace Reference . . . . .	61
5.9	CEGUI::ItemListboxProperties Namespace Reference . . . . .	62
5.10	CEGUI::ListboxProperties Namespace Reference . . . . .	63
5.11	CEGUI::ListHeaderProperties Namespace Reference . . . . .	64
5.12	CEGUI::ListHeaderSegmentProperties Namespace Reference . . . . .	65
5.13	CEGUI::MultiColumnListProperties Namespace Reference . . . . .	66
5.14	CEGUI::MultiLineEditboxProperties Namespace Reference . . . . .	67
5.15	CEGUI::ProgressBarProperties Namespace Reference . . . . .	68
5.16	CEGUI::RadioButtonProperties Namespace Reference . . . . .	69
5.17	CEGUI::ScrollablePaneProperties Namespace Reference . . . . .	70

5.18	CEGUI::ScrollbarProperties Namespace Reference	71
5.19	CEGUI::ScrolledContainerProperties Namespace Reference	72
5.20	CEGUI::ScrolledItemListBaseProperties Namespace Reference	73
5.21	CEGUI::SliderProperties Namespace Reference	74
5.22	CEGUI::TabControlProperties Namespace Reference	75
5.23	CEGUI::ThumbProperties Namespace Reference	76
5.24	CEGUI::TitlebarProperties Namespace Reference	77
5.25	CEGUI::TooltipProperties Namespace Reference	78
5.26	CEGUI::TreeProperties Namespace Reference	79
5.27	CEGUI::WindowProperties Namespace Reference	80
<b>6</b>	<b>Crazy Eddies GUI System Class Documentation</b>	<b>83</b>
6.1	CEGUI::AbsoluteDim Class Reference	83
6.2	CEGUI::ActivationEventArgs Class Reference	85
6.3	CEGUI::EditboxProperties::ActiveSelectionColour Class Reference	86
6.4	CEGUI::MenuBaseProperties::AllowMultiplePopups Class Reference	88
6.5	CEGUI::WindowProperties::Alpha Class Reference	90
6.6	CEGUI::AlreadyExistsException Class Reference	92
6.7	CEGUI::WindowProperties::AlwaysOnTop Class Reference	94
6.8	CEGUI::WindowProperties::AutoRepeatDelay Class Reference	96
6.9	CEGUI::WindowProperties::AutoRepeatRate Class Reference	98
6.10	CEGUI::ItemListBaseProperties::AutoResizeEnabled Class Reference	100
6.11	CEGUI::BaseDim Class Reference	102
6.12	CEGUI::BoundSlot Class Reference	106
6.13	CEGUI::ButtonBase Class Reference	109
6.14	CEGUI::MultiLineEditboxProperties::CaratIndex Class Reference	113
6.15	CEGUI::ComboboxProperties::CaratIndex Class Reference	115
6.16	CEGUI::EditboxProperties::CaratIndex Class Reference	117
6.17	CEGUI::Checkbox Class Reference	119
6.18	CEGUI::ScrolledContainerProperties::ChildExtentsArea Class Reference	122
6.19	CEGUI::ListHeaderSegmentProperties::Clickable Class Reference	124
6.20	CEGUI::SliderProperties::ClickStepSize Class Reference	126
6.21	CEGUI::WindowProperties::ClippedByParent Class Reference	128
6.22	CEGUI::ClippedContainer Class Reference	130
6.23	CEGUI::FrameWindowProperties::CloseButtonEnabled Class Reference	135
6.24	CEGUI::colour Class Reference	137
6.25	CEGUI::ColourRect Class Reference	138



6.26	CEGUI::MultiColumnListProperties::ColumnHeader Class Reference . . . . .	142
6.27	CEGUI::ListHeaderProperties::ColumnsMovable Class Reference . . . . .	144
6.28	CEGUI::MultiColumnListProperties::ColumnsMovable Class Reference . . . . .	146
6.29	CEGUI::ListHeaderProperties::ColumnsSizable Class Reference . . . . .	148
6.30	CEGUI::MultiColumnListProperties::ColumnsSizable Class Reference . . . . .	150
6.31	CEGUI::Combobox Class Reference . . . . .	152
6.32	CEGUI::ComboDropList Class Reference . . . . .	171
6.33	CEGUI::ComponentArea Class Reference . . . . .	177
6.34	CEGUI::Config_xmlHandler Class Reference . . . . .	180
6.35	CEGUI::ConstBaseIterator< T > Class Template Reference . . . . .	182
6.36	CEGUI::ScrollablePaneProperties::ContentArea Class Reference . . . . .	185
6.37	CEGUI::ScrolledContainerProperties::ContentArea Class Reference . . . . .	187
6.38	CEGUI::ScrollablePaneProperties::ContentPaneAutoSized Class Reference . . . . .	189
6.39	CEGUI::ScrolledContainerProperties::ContentPaneAutoSized Class Reference . . . . .	191
6.40	CEGUI::CoordConverter Class Reference . . . . .	193
6.41	CEGUI::ProgressBarProperties::CurrentProgress Class Reference . . . . .	199
6.42	CEGUI::SliderProperties::CurrentValue Class Reference . . . . .	201
6.43	CEGUI::SpinnerProperties::CurrentValue Class Reference . . . . .	203
6.44	CEGUI::WindowProperties::CustomTooltipType Class Reference . . . . .	205
6.45	CEGUI::DefaultLogger Class Reference . . . . .	207
6.46	CEGUI::WindowProperties::DestroyedByParent Class Reference . . . . .	209
6.47	CEGUI::Dimension Class Reference . . . . .	211
6.48	CEGUI::WindowProperties::Disabled Class Reference . . . . .	214
6.49	CEGUI::TooltipProperties::DisplayTime Class Reference . . . . .	216
6.50	CEGUI::WindowProperties::DistributeCapturedInputs Class Reference . . . . .	218
6.51	CEGUI::ScrollbarProperties::DocumentSize Class Reference . . . . .	220
6.52	CEGUI::ListHeaderSegmentProperties::Dragable Class Reference . . . . .	222
6.53	CEGUI::DragContainerProperties::DragAlpha Class Reference . . . . .	224
6.54	CEGUI::DragContainer Class Reference . . . . .	226
6.55	CEGUI::DragContainerProperties::DragCursorImage Class Reference . . . . .	239
6.56	CEGUI::DragDropEventArgs Class Reference . . . . .	241
6.57	CEGUI::WindowProperties::DragDropTarget Class Reference . . . . .	242
6.58	CEGUI::TitlebarProperties::DraggingEnabled Class Reference . . . . .	244
6.59	CEGUI::DragContainerProperties::DraggingEnabled Class Reference . . . . .	246
6.60	CEGUI::FrameWindowProperties::DragMovingEnabled Class Reference . . . . .	248
6.61	CEGUI::DragContainerProperties::DragThreshold Class Reference . . . . .	250

6.62	CEGUI::DynamicModule Class Reference . . . . .	252
6.63	CEGUI::Editbox Class Reference . . . . .	254
6.64	CEGUI::EditboxWindowRenderer Class Reference . . . . .	267
6.65	CEGUI::ComboboxProperties::EditSelectionLength Class Reference . . . . .	270
6.66	CEGUI::ComboboxProperties::EditSelectionStart Class Reference . . . . .	272
6.67	CEGUI::Event Class Reference . . . . .	274
6.68	CEGUI::Event::ScopedConnection Class Reference . . . . .	277
6.69	CEGUI::EventArgs Class Reference . . . . .	278
6.70	CEGUI::EventSet Class Reference . . . . .	279
6.71	CEGUI::FrameWindowProperties::EWSizingCursorImage Class Reference . . . . .	285
6.72	CEGUI::Exception Class Reference . . . . .	287
6.73	CEGUI::FactoryModule Class Reference . . . . .	290
6.74	CEGUI::PopupMenuProperties::FadeInTime Class Reference . . . . .	292
6.75	CEGUI::PopupMenuProperties::FadeOutTime Class Reference . . . . .	294
6.76	CEGUI::TooltipProperties::FadeTime Class Reference . . . . .	296
6.77	CEGUI::Falagard_xmlHandler Class Reference . . . . .	298
6.78	CEGUI::FalagardComponentBase Class Reference . . . . .	299
6.79	CEGUI::FalagardXMLHelper Class Reference . . . . .	306
6.80	CEGUI::FileIOException Class Reference . . . . .	307
6.81	CEGUI::WindowProperties::Font Class Reference . . . . .	309
6.82	CEGUI::Font Class Reference . . . . .	311
6.83	CEGUI::Font_xmlHandler Class Reference . . . . .	326
6.84	CEGUI::FontDim Class Reference . . . . .	327
6.85	CEGUI::FontGlyph Class Reference . . . . .	329
6.86	CEGUI::FontManager Class Reference . . . . .	331
6.87	CEGUI::ScrollablePaneProperties::ForceHorzScrollbar Class Reference . . . . .	336
6.88	CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar Class Reference . . . . .	338
6.89	CEGUI::ComboboxProperties::ForceHorzScrollbar Class Reference . . . . .	340
6.90	CEGUI::TreeProperties::ForceHorzScrollbar Class Reference . . . . .	342
6.91	CEGUI::ListboxProperties::ForceHorzScrollbar Class Reference . . . . .	344
6.92	CEGUI::MultiColumnListProperties::ForceHorzScrollbar Class Reference . . . . .	346
6.93	CEGUI::MultiLineEditboxProperties::ForceVertScrollbar Class Reference . . . . .	348
6.94	CEGUI::ScrollablePaneProperties::ForceVertScrollbar Class Reference . . . . .	350
6.95	CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar Class Reference . . . . .	352
6.96	CEGUI::ComboboxProperties::ForceVertScrollbar Class Reference . . . . .	354
6.97	CEGUI::ListboxProperties::ForceVertScrollbar Class Reference . . . . .	356

6.98 CEGUI::TreeProperties::ForceVertScrollbar Class Reference . . . . .	358
6.99 CEGUI::MultiColumnListProperties::ForceVertScrollbar Class Reference . . . . .	360
6.100CEGUI::FrameComponent Class Reference . . . . .	362
6.101CEGUI::FrameWindowProperties::FrameEnabled Class Reference . . . . .	366
6.102CEGUI::FrameWindow Class Reference . . . . .	368
6.103CEGUI::FreeFunctionSlot Class Reference . . . . .	386
6.104CEGUI::FreeTypeFont Class Reference . . . . .	387
6.105CEGUI::FunctorCopySlot< T > Class Template Reference . . . . .	391
6.106CEGUI::FunctorPointerSlot< T > Class Template Reference . . . . .	392
6.107CEGUI::FunctorReferenceBinder< T > Struct Template Reference . . . . .	393
6.108CEGUI::FunctorReferenceSlot< T > Class Template Reference . . . . .	394
6.109CEGUI::GenericException Class Reference . . . . .	395
6.110CEGUI::GlobalEventSet Class Reference . . . . .	397
6.111CEGUI::GroupBox Class Reference . . . . .	399
6.112CEGUI::RadioButtonProperties::GroupID Class Reference . . . . .	403
6.113CEGUI::GUILayout_xmlHandler Class Reference . . . . .	405
6.114CEGUI::GUISheet Class Reference . . . . .	408
6.115CEGUI::HeaderSequenceEventArgs Class Reference . . . . .	411
6.116CEGUI::WindowProperties::HorizontalAlignment Class Reference . . . . .	412
6.117CEGUI::ThumbProperties::HorzFree Class Reference . . . . .	414
6.118CEGUI::ScrollablePaneProperties::HorzOverlapSize Class Reference . . . . .	416
6.119CEGUI::ThumbProperties::HorzRange Class Reference . . . . .	418
6.120CEGUI::ScrollablePaneProperties::HorzScrollPosition Class Reference . . . . .	420
6.121CEGUI::ScrollablePaneProperties::HorzStepSize Class Reference . . . . .	422
6.122CEGUI::ThumbProperties::HotTracked Class Reference . . . . .	424
6.123CEGUI::TooltipProperties::HoverTime Class Reference . . . . .	426
6.124CEGUI::WindowProperties::ID Class Reference . . . . .	428
6.125CEGUI::Image Class Reference . . . . .	430
6.126CEGUI::ImageCodec Class Reference . . . . .	439
6.127CEGUI::ImageDim Class Reference . . . . .	441
6.128CEGUI::ImageryComponent Class Reference . . . . .	443
6.129CEGUI::ImagerySection Class Reference . . . . .	448
6.130CEGUI::Imageset Class Reference . . . . .	454
6.131CEGUI::Imageset_xmlHandler Class Reference . . . . .	466
6.132CEGUI::ImagesetManager Class Reference . . . . .	468
6.133CEGUI::EditboxProperties::InactiveSelectionColour Class Reference . . . . .	472

6.134CEGUI::WindowProperties::InheritsAlpha Class Reference . . . . .	474
6.135CEGUI::WindowProperties::InheritsTooltipText Class Reference . . . . .	476
6.136CEGUI::InvalidRequestException Class Reference . . . . .	478
6.137CEGUI::ItemEntry Class Reference . . . . .	480
6.138CEGUI::ItemEntryWindowRenderer Class Reference . . . . .	486
6.139CEGUI::ItemListBase Class Reference . . . . .	489
6.140CEGUI::ItemListBaseWindowRenderer Class Reference . . . . .	501
6.141CEGUI::ItemListbox Class Reference . . . . .	504
6.142CEGUI::MenuBaseProperties::ItemSpacing Class Reference . . . . .	509
6.143CEGUI::TreeProperties::ItemTooltips Class Reference . . . . .	511
6.144CEGUI::ListboxProperties::ItemTooltips Class Reference . . . . .	513
6.145CEGUI::Key Struct Reference . . . . .	515
6.146CEGUI::KeyEventArgs Class Reference . . . . .	517
6.147CEGUI::LayerSpecification Class Reference . . . . .	518
6.148CEGUI::Listbox Class Reference . . . . .	521
6.149CEGUI::ListboxItem Class Reference . . . . .	537
6.150CEGUI::ListboxTextItem Class Reference . . . . .	546
6.151CEGUI::ListboxWindowRenderer Class Reference . . . . .	550
6.152CEGUI::ListHeader Class Reference . . . . .	553
6.153CEGUI::ListHeaderSegment Class Reference . . . . .	571
6.154CEGUI::ListHeaderWindowRenderer Class Reference . . . . .	581
6.155CEGUI::Logger Class Reference . . . . .	584
6.156CEGUI::WindowProperties::LookNFeel Class Reference . . . . .	587
6.157CEGUI::EditboxProperties::MaskCodepoint Class Reference . . . . .	589
6.158CEGUI::EditboxProperties::MaskText Class Reference . . . . .	591
6.159CEGUI::ComboboxProperties::MaxEditTextLength Class Reference . . . . .	593
6.160CEGUI::SliderProperties::MaximumValue Class Reference . . . . .	595
6.161CEGUI::SpinnerProperties::MaximumValue Class Reference . . . . .	597
6.162CEGUI::MultiLineEditboxProperties::MaxTextLength Class Reference . . . . .	599
6.163CEGUI::EditboxProperties::MaxTextLength Class Reference . . . . .	601
6.164CEGUI::MCLGridRef Struct Reference . . . . .	603
6.165CEGUI::MemberFunctionSlot< T > Class Template Reference . . . . .	604
6.166CEGUI::MemoryException Class Reference . . . . .	605
6.167CEGUI::Menubar Class Reference . . . . .	607
6.168CEGUI::MenuBase Class Reference . . . . .	610
6.169CEGUI::MenuItem Class Reference . . . . .	613

6.170CEGUI::SpinnerProperties::MinimumValue Class Reference . . . . .	620
6.171CEGUI::WindowProperties::MouseButtonDownAutoRepeat Class Reference . . . . .	622
6.172CEGUI::MouseClickedTracker Struct Reference . . . . .	624
6.173CEGUI::MouseCursor Class Reference . . . . .	625
6.174CEGUI::MouseCursorEventArgs Class Reference . . . . .	631
6.175CEGUI::WindowProperties::MouseCursorImage Class Reference . . . . .	632
6.176CEGUI::MouseEventArgs Class Reference . . . . .	634
6.177CEGUI::WindowProperties::MousePassThroughEnabled Class Reference . . . . .	636
6.178CEGUI::ListHeaderSegmentProperties::MovingCursorImage Class Reference . . . . .	638
6.179CEGUI::MultiColumnList Class Reference . . . . .	640
6.180CEGUI::MultiColumnList::ListRow Struct Reference . . . . .	674
6.181CEGUI::MultiColumnListWindowRenderer Class Reference . . . . .	675
6.182CEGUI::MultiLineEditbox Class Reference . . . . .	678
6.183CEGUI::MultiLineEditbox::LineInfo Struct Reference . . . . .	693
6.184CEGUI::MultiLineEditboxWindowRenderer Class Reference . . . . .	694
6.185CEGUI::TreeProperties::MultiSelect Class Reference . . . . .	697
6.186CEGUI::ItemListboxProperties::MultiSelect Class Reference . . . . .	699
6.187CEGUI::ListboxProperties::MultiSelect Class Reference . . . . .	701
6.188CEGUI::NamedArea Class Reference . . . . .	703
6.189CEGUI::FrameWindowProperties::NESWSizingCursorImage Class Reference . . . . .	705
6.190CEGUI::MultiColumnListProperties::NominatedSelectionColumnID Class Reference . . . . .	707
6.191CEGUI::MultiColumnListProperties::NominatedSelectionRow Class Reference . . . . .	709
6.192CEGUI::EditboxProperties::NormalTextColour Class Reference . . . . .	711
6.193CEGUI::FrameWindowProperties::NSSizingCursorImage Class Reference . . . . .	713
6.194CEGUI::NullObjectException Class Reference . . . . .	715
6.195CEGUI::FrameWindowProperties::NWSEizingCursorImage Class Reference . . . . .	717
6.196CEGUI::ObjectInUseException Class Reference . . . . .	719
6.197CEGUI::ScrollbarProperties::OverlapSize Class Reference . . . . .	721
6.198CEGUI::ScrollbarProperties::PageSize Class Reference . . . . .	723
6.199CEGUI::PixmapFont Class Reference . . . . .	725
6.200CEGUI::PopupMenu Class Reference . . . . .	728
6.201CEGUI::ProgressBar Class Reference . . . . .	734
6.202CEGUI::Property Class Reference . . . . .	739
6.203CEGUI::PropertyDefinition Class Reference . . . . .	748
6.204CEGUI::PropertyDefinitionBase Class Reference . . . . .	751
6.205CEGUI::PropertyDim Class Reference . . . . .	754

6.206CEGUI::PropertyHelper Class Reference . . . . .	756
6.207CEGUI::PropertyInitialiser Class Reference . . . . .	757
6.208CEGUI::PropertyLinkDefinition Class Reference . . . . .	759
6.209CEGUI::PropertyReceiver Class Reference . . . . .	762
6.210CEGUI::PropertySet Class Reference . . . . .	763
6.211CEGUI::PushButton Class Reference . . . . .	767
6.212CEGUI::RadioButton Class Reference . . . . .	770
6.213CEGUI::RawDataContainer Class Reference . . . . .	774
6.214CEGUI::EditboxProperties::ReadOnly Class Reference . . . . .	776
6.215CEGUI::ComboboxProperties::ReadOnly Class Reference . . . . .	778
6.216CEGUI::MultiLineEditboxProperties::ReadOnly Class Reference . . . . .	780
6.217CEGUI::Rect Class Reference . . . . .	782
6.218CEGUI::RefCounted< T > Class Template Reference . . . . .	785
6.219CEGUI::RegexValidator Struct Reference . . . . .	787
6.220CEGUI::RenderCache Class Reference . . . . .	788
6.221CEGUI::Renderer Class Reference . . . . .	791
6.222CEGUI::RendererException Class Reference . . . . .	799
6.223CEGUI::ResourceProvider Class Reference . . . . .	801
6.224CEGUI::WindowProperties::RestoreOldCapture Class Reference . . . . .	804
6.225CEGUI::WindowProperties::RiseOnClick Class Reference . . . . .	806
6.226CEGUI::FrameWindowProperties::RollUpEnabled Class Reference . . . . .	808
6.227CEGUI::FrameWindowProperties::RollUpState Class Reference . . . . .	810
6.228CEGUI::MultiColumnListProperties::RowCount Class Reference . . . . .	812
6.229CEGUI::Scheme Class Reference . . . . .	814
6.230CEGUI::Scheme_xmlHandler Class Reference . . . . .	817
6.231CEGUI::SchemeManager Class Reference . . . . .	819
6.232CEGUI::ScriptException Class Reference . . . . .	822
6.233CEGUI::ScriptFunctor Class Reference . . . . .	824
6.234CEGUI::ScriptModule Class Reference . . . . .	825
6.235CEGUI::ScriptWindowHelper Class Reference . . . . .	830
6.236CEGUI::ScrollablePane Class Reference . . . . .	832
6.237CEGUI::ScrollablePaneWindowRenderer Class Reference . . . . .	846
6.238CEGUI::Scrollbar Class Reference . . . . .	849
6.239CEGUI::ScrollbarWindowRenderer Class Reference . . . . .	859
6.240CEGUI::ScrolledContainer Class Reference . . . . .	862
6.241CEGUI::ScrolledItemListBase Class Reference . . . . .	868

6.242CEGUI::ScrollbarProperties::ScrollPosition Class Reference . . . . .	871
6.243CEGUI::SectionSpecification Class Reference . . . . .	873
6.244CEGUI::ItemEntryProperties::Selectable Class Reference . . . . .	879
6.245CEGUI::RadioButtonProperties::Selected Class Reference . . . . .	881
6.246CEGUI::ItemEntryProperties::Selected Class Reference . . . . .	883
6.247CEGUI::CheckboxProperties::Selected Class Reference . . . . .	885
6.248CEGUI::EditboxProperties::SelectedTextColour Class Reference . . . . .	887
6.249CEGUI::MultiLineEditboxProperties::SelectionBrushImage Class Reference . . . . .	889
6.250CEGUI::MultiLineEditboxProperties::SelectionLength Class Reference . . . . .	891
6.251CEGUI::EditboxProperties::SelectionLength Class Reference . . . . .	893
6.252CEGUI::MultiColumnListProperties::SelectionMode Class Reference . . . . .	895
6.253CEGUI::EditboxProperties::SelectionStart Class Reference . . . . .	897
6.254CEGUI::MultiLineEditboxProperties::SelectionStart Class Reference . . . . .	899
6.255CEGUI::SimpleTimer Class Reference . . . . .	901
6.256CEGUI::ComboboxProperties::SingleClickMode Class Reference . . . . .	902
6.257CEGUI::ListHeaderSegmentProperties::Sizable Class Reference . . . . .	904
6.258CEGUI::Size Class Reference . . . . .	906
6.259CEGUI::FrameWindowProperties::SizingBorderThickness Class Reference . . . . .	907
6.260CEGUI::ListHeaderSegmentProperties::SizingCursorImage Class Reference . . . . .	909
6.261CEGUI::FrameWindowProperties::SizingEnabled Class Reference . . . . .	911
6.262CEGUI::Slider Class Reference . . . . .	913
6.263CEGUI::SliderWindowRenderer Class Reference . . . . .	920
6.264CEGUI::SlotFunctorBase Class Reference . . . . .	923
6.265CEGUI::TreeProperties::Sort Class Reference . . . . .	924
6.266CEGUI::ListboxProperties::Sort Class Reference . . . . .	926
6.267CEGUI::MultiColumnListProperties::SortColumnID Class Reference . . . . .	928
6.268CEGUI::ListHeaderProperties::SortColumnID Class Reference . . . . .	930
6.269CEGUI::ListHeaderSegmentProperties::SortDirection Class Reference . . . . .	932
6.270CEGUI::ListHeaderProperties::SortDirection Class Reference . . . . .	934
6.271CEGUI::MultiColumnListProperties::SortDirection Class Reference . . . . .	936
6.272CEGUI::ItemListBaseProperties::SortEnabled Class Reference . . . . .	938
6.273CEGUI::ComboboxProperties::SortList Class Reference . . . . .	940
6.274CEGUI::ItemListBaseProperties::SortMode Class Reference . . . . .	942
6.275CEGUI::MultiColumnListProperties::SortSettingEnabled Class Reference . . . . .	944
6.276CEGUI::ListHeaderProperties::SortSettingEnabled Class Reference . . . . .	946
6.277CEGUI::Spinner Class Reference . . . . .	948

6.278CEGUI::StateImagery Class Reference . . . . .	959
6.279CEGUI::ProgressBarProperties::StepSize Class Reference . . . . .	962
6.280CEGUI::SpinnerProperties::StepSize Class Reference . . . . .	964
6.281CEGUI::ScrollbarProperties::StepSize Class Reference . . . . .	966
6.282CEGUI::String Class Reference . . . . .	968
6.283CEGUI::String::const_iterator Class Reference . . . . .	1035
6.284CEGUI::String::FastLessCompare Struct Reference . . . . .	1036
6.285CEGUI::String::iterator Class Reference . . . . .	1037
6.286CEGUI::SubComp Class Reference . . . . .	1038
6.287CEGUI::SubscriberSlot Class Reference . . . . .	1039
6.288CEGUI::System Class Reference . . . . .	1041
6.289CEGUI::TabButton Class Reference . . . . .	1058
6.290CEGUI::TabControl Class Reference . . . . .	1062
6.291CEGUI::TabControlWindowRenderer Class Reference . . . . .	1073
6.292CEGUI::TabControlProperties::TabHeight Class Reference . . . . .	1076
6.293CEGUI::TabControlProperties::TabPagePosition Class Reference . . . . .	1078
6.294CEGUI::TabControlProperties::TabTextPadding Class Reference . . . . .	1080
6.295CEGUI::WindowProperties::Text Class Reference . . . . .	1082
6.296CEGUI::TextComponent Class Reference . . . . .	1084
6.297CEGUI::SpinnerProperties::TextInputMode Class Reference . . . . .	1089
6.298CEGUI::Texture Class Reference . . . . .	1091
6.299CEGUI::TextUtils Class Reference . . . . .	1095
6.300CEGUI::Thumb Class Reference . . . . .	1098
6.301CEGUI::Titlebar Class Reference . . . . .	1104
6.302CEGUI::FrameWindowProperties::TitlebarEnabled Class Reference . . . . .	1110
6.303CEGUI::WindowProperties::Tooltip Class Reference . . . . .	1112
6.304CEGUI::Tooltip Class Reference . . . . .	1114
6.305CEGUI::TooltipWindowRenderer Class Reference . . . . .	1123
6.306CEGUI::Tree Class Reference . . . . .	1126
6.307CEGUI::TreeEventArgs Class Reference . . . . .	1143
6.308CEGUI::TreeItem Class Reference . . . . .	1144
6.309CEGUI::UDim Class Reference . . . . .	1156
6.310CEGUI::WindowProperties::UnifiedAreaRect Class Reference . . . . .	1157
6.311CEGUI::UnifiedDim Class Reference . . . . .	1159
6.312CEGUI::WindowProperties::UnifiedHeight Class Reference . . . . .	1161
6.313CEGUI::WindowProperties::UnifiedMaxSize Class Reference . . . . .	1163



6.314CEGUI::WindowProperties::UnifiedMinSize Class Reference . . . . .	1165
6.315CEGUI::WindowProperties::UnifiedPosition Class Reference . . . . .	1167
6.316CEGUI::WindowProperties::UnifiedSize Class Reference . . . . .	1169
6.317CEGUI::WindowProperties::UnifiedWidth Class Reference . . . . .	1171
6.318CEGUI::WindowProperties::UnifiedXPosition Class Reference . . . . .	1173
6.319CEGUI::WindowProperties::UnifiedYPosition Class Reference . . . . .	1175
6.320CEGUI::UnknownObjectException Class Reference . . . . .	1177
6.321CEGUI::UpdateEventArgs Class Reference . . . . .	1179
6.322CEGUI::URect Class Reference . . . . .	1180
6.323CEGUI::UVector2 Class Reference . . . . .	1181
6.324CEGUI::ComboboxProperties::ValidationString Class Reference . . . . .	1182
6.325CEGUI::EditboxProperties::ValidationString Class Reference . . . . .	1184
6.326CEGUI::Vector2 Class Reference . . . . .	1186
6.327CEGUI::Vector3 Class Reference . . . . .	1187
6.328CEGUI::ThumbProperties::VertFree Class Reference . . . . .	1188
6.329CEGUI::WindowProperties::VerticalAlignment Class Reference . . . . .	1190
6.330CEGUI::ScrollablePaneProperties::VertOverlapSize Class Reference . . . . .	1192
6.331CEGUI::ThumbProperties::VertRange Class Reference . . . . .	1194
6.332CEGUI::ScrollablePaneProperties::VertScrollPosition Class Reference . . . . .	1196
6.333CEGUI::ScrollablePaneProperties::VertStepSize Class Reference . . . . .	1198
6.334CEGUI::WindowProperties::Visible Class Reference . . . . .	1200
6.335CEGUI::WindowProperties::WantsMultiClickEvents Class Reference . . . . .	1202
6.336CEGUI::WidgetComponent Class Reference . . . . .	1204
6.337CEGUI::WidgetDim Class Reference . . . . .	1206
6.338CEGUI::WidgetLookFeel Class Reference . . . . .	1208
6.339CEGUI::WidgetLookManager Class Reference . . . . .	1217
6.340CEGUI::Window Class Reference . . . . .	1222
6.341CEGUI::WindowEventArgs Class Reference . . . . .	1298
6.342CEGUI::WindowFactory Class Reference . . . . .	1299
6.343CEGUI::WindowFactoryManager Class Reference . . . . .	1301
6.344CEGUI::WindowFactoryManager::AliasTargetStack Class Reference . . . . .	1308
6.345CEGUI::WindowFactoryManager::FalagardWindowMapping Struct Reference . . . . .	1309
6.346CEGUI::WindowManager Class Reference . . . . .	1310
6.347CEGUI::WindowRenderer Class Reference . . . . .	1318
6.348CEGUI::WindowProperties::WindowRenderer Class Reference . . . . .	1323
6.349CEGUI::WindowRendererFactory Class Reference . . . . .	1325

---

6.350CEGUI::MultiLineEditboxProperties::WordWrap Class Reference . . . . .	1326
6.351CEGUI::XMLAttributes Class Reference . . . . .	1328
6.352CEGUI::XMLParser Class Reference . . . . .	1333
6.353CEGUI::XMLSerializer Class Reference . . . . .	1336
6.354CEGUI::WindowProperties::ZOrderChangeEnabled Class Reference . . . . .	1339
<b>7 Crazy Eddies GUI System Page Documentation</b>	<b>1341</b>
7.1 Todo List . . . . .	1341
7.2 Bug List . . . . .	1342

# Chapter 1

## Crazy Eddies GUI System Namespace Index

### 1.1 Crazy Eddies GUI System Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">CEGUI</a> (Main namespace for Crazy Eddie's GUI Library ) . . . . .	25
<a href="#">CEGUI::CheckboxProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Checkbox</a> class ) . . . . .	55
<a href="#">CEGUI::ComboboxProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Combobox</a> class ) . . . . .	56
<a href="#">CEGUI::EditboxProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Editbox</a> class ) . . . . .	57
<a href="#">CEGUI::FontProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Font</a> base class ) . . . . .	58
<a href="#">CEGUI::FrameWindowProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">FrameWindow</a> class ) . . . . .	59
<a href="#">CEGUI::ItemEntryProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ItemEntry</a> class ) . . . . .	60
<a href="#">CEGUI::ItemListBaseProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ItemListBase</a> class ) . . . . .	61
<a href="#">CEGUI::ItemListboxProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ItemListbox</a> class ) . . . . .	62
<a href="#">CEGUI::ListboxProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Listbox</a> class ) . . . . .	63
<a href="#">CEGUI::ListHeaderProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ListHeader</a> class ) . . . . .	64
<a href="#">CEGUI::ListHeaderSegmentProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ListHeaderSegment</a> class ) . . . . .	65
<a href="#">CEGUI::MultiColumnListProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">MultiColumnList</a> class ) . . . . .	66
<a href="#">CEGUI::MultiLineEditboxProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">MultiLineEditbox</a> class ) . . . . .	67
<a href="#">CEGUI::ProgressBarProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ProgressBar</a> class ) . . . . .	68
<a href="#">CEGUI::RadioButtonProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">RadioButton</a> class ) . . . . .	69

<a href="#">CEGUI::ScrollablePaneProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ScrollablePane</a> class ) . . . . .	70
<a href="#">CEGUI::ScrollbarProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Scrollbar</a> class ) . . . . .	71
<a href="#">CEGUI::ScrolledContainerProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ScrolledContainer</a> class ) . . . . .	72
<a href="#">CEGUI::ScrolledItemListBaseProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">ScrolledItemListBase</a> class ) . . . . .	73
<a href="#">CEGUI::SliderProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Slider</a> class ) . . . . .	74
<a href="#">CEGUI::TabControlProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Listbox</a> class ) . . . . .	75
<a href="#">CEGUI::ThumbProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Thumb</a> class ) . . . . .	76
<a href="#">CEGUI::TitlebarProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Titlebar</a> class ) . . . . .	77
<a href="#">CEGUI::TooltipProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Tooltip</a> class ) . . . . .	78
<a href="#">CEGUI::TreeProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Listbox</a> class ) . . . . .	79
<a href="#">CEGUI::WindowProperties</a> (Namespace containing all classes that make up the properties interface for the <a href="#">Window</a> base class ) . . . . .	80

## Chapter 2

# Crazy Eddies GUI System Hierarchical Index

### 2.1 Crazy Eddies GUI System Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

CEGUI::BaseDim . . . . .	102
CEGUI::AbsoluteDim . . . . .	83
CEGUI::FontDim . . . . .	327
CEGUI::ImageDim . . . . .	441
CEGUI::PropertyDim . . . . .	754
CEGUI::UnifiedDim . . . . .	1159
CEGUI::WidgetDim . . . . .	1206
CEGUI::BoundSlot . . . . .	106
CEGUI::colour . . . . .	137
CEGUI::ColourRect . . . . .	138
CEGUI::ComponentArea . . . . .	177
CEGUI::Config_xmlHandler . . . . .	180
CEGUI::ConstBaseIterator< T > . . . . .	182
CEGUI::CoordConverter . . . . .	193
CEGUI::Dimension . . . . .	211
CEGUI::DynamicModule . . . . .	252
CEGUI::Event . . . . .	274
CEGUI::EventArgs . . . . .	278
CEGUI::MouseEventArgs . . . . .	631
CEGUI::WindowEventArgs . . . . .	1298
CEGUI::ActivationEventArgs . . . . .	85
CEGUI::DragDropEventArgs . . . . .	241
CEGUI::HeaderSequenceEventArgs . . . . .	411
CEGUI::KeyEventArgs . . . . .	517
CEGUI::MouseEventArgs . . . . .	634
CEGUI::TreeEventArgs . . . . .	1143
CEGUI::UpdateEventArgs . . . . .	1179
CEGUI::EventSet . . . . .	279
CEGUI::GlobalEventSet . . . . .	397
CEGUI::MouseCursor . . . . .	625

CEGUI::Renderer . . . . .	791
CEGUI::System . . . . .	1041
CEGUI::Window . . . . .	1222
CEGUI::ButtonBase . . . . .	109
CEGUI::Checkbox . . . . .	119
CEGUI::PushButton . . . . .	767
CEGUI::Thumb . . . . .	1098
CEGUI::RadioButton . . . . .	770
CEGUI::TabButton . . . . .	1058
CEGUI::ClippedContainer . . . . .	130
CEGUI::Combobox . . . . .	152
CEGUI::DragContainer . . . . .	226
CEGUI::Editbox . . . . .	254
CEGUI::FrameWindow . . . . .	368
CEGUI::GroupBox . . . . .	399
CEGUI::GUISheet . . . . .	408
CEGUI::ItemEntry . . . . .	480
CEGUI::MenuItem . . . . .	613
CEGUI::ItemListBase . . . . .	489
CEGUI::MenuBase . . . . .	610
CEGUI::Menubar . . . . .	607
CEGUI::PopupMenu . . . . .	728
CEGUI::ScrolledItemListBase . . . . .	868
CEGUI::ItemListbox . . . . .	504
CEGUI::Listbox . . . . .	521
CEGUI::ComboDropList . . . . .	171
CEGUI::ListHeader . . . . .	553
CEGUI::ListHeaderSegment . . . . .	571
CEGUI::MultiColumnList . . . . .	640
CEGUI::MultiLineEditbox . . . . .	678
CEGUI::ProgressBar . . . . .	734
CEGUI::ScrollablePane . . . . .	832
CEGUI::Scrollbar . . . . .	849
CEGUI::ScrolledContainer . . . . .	862
CEGUI::Slider . . . . .	913
CEGUI::Spinner . . . . .	948
CEGUI::TabControl . . . . .	1062
CEGUI::Titlebar . . . . .	1104
CEGUI::Tooltip . . . . .	1114
CEGUI::Tree . . . . .	1126
CEGUI::Exception . . . . .	287
CEGUI::AlreadyExistsException . . . . .	92
CEGUI::FileIOException . . . . .	307
CEGUI::GenericException . . . . .	395
CEGUI::InvalidRequestException . . . . .	478
CEGUI::MemoryException . . . . .	605
CEGUI::NullObjectException . . . . .	715
CEGUI::ObjectInUseException . . . . .	719
CEGUI::RendererException . . . . .	799
CEGUI::ScriptException . . . . .	822
CEGUI::UnknownObjectException . . . . .	1177
CEGUI::FactoryModule . . . . .	290

CEGUI::Falagard_xmlHandler . . . . .	298
CEGUI::FalagardComponentBase . . . . .	299
CEGUI::FrameComponent . . . . .	362
CEGUI::ImageryComponent . . . . .	443
CEGUI::TextComponent . . . . .	1084
CEGUI::FalagardXMLHelper . . . . .	306
CEGUI::Font_xmlHandler . . . . .	326
CEGUI::FontGlyph . . . . .	329
CEGUI::FontManager . . . . .	331
CEGUI::FunctorReferenceBinder< T > . . . . .	393
CEGUI::GUILayout_xmlHandler . . . . .	405
CEGUI::Image . . . . .	430
CEGUI::ImageCodec . . . . .	439
CEGUI::ImagerySection . . . . .	448
CEGUI::Imageset . . . . .	454
CEGUI::Imageset_xmlHandler . . . . .	466
CEGUI::ImagesetManager . . . . .	468
CEGUI::Key . . . . .	515
CEGUI::LayerSpecification . . . . .	518
CEGUI::ListboxItem . . . . .	537
CEGUI::ListboxTextItem . . . . .	546
CEGUI::Logger . . . . .	584
CEGUI::DefaultLogger . . . . .	207
CEGUI::MCLGridRef . . . . .	603
CEGUI::MouseClickedTracker . . . . .	624
CEGUI::MultiColumnList::ListRow . . . . .	674
CEGUI::MultiLineEditbox::LineInfo . . . . .	693
CEGUI::NamedArea . . . . .	703
CEGUI::Property . . . . .	739
CEGUI::CheckboxProperties::Selected . . . . .	885
CEGUI::ComboboxProperties::CaratIndex . . . . .	115
CEGUI::ComboboxProperties::EditSelectionLength . . . . .	270
CEGUI::ComboboxProperties::EditSelectionStart . . . . .	272
CEGUI::ComboboxProperties::ForceHorzScrollbar . . . . .	340
CEGUI::ComboboxProperties::ForceVertScrollbar . . . . .	354
CEGUI::ComboboxProperties::MaxEditTextLength . . . . .	593
CEGUI::ComboboxProperties::ReadOnly . . . . .	778
CEGUI::ComboboxProperties::SingleClickMode . . . . .	902
CEGUI::ComboboxProperties::SortList . . . . .	940
CEGUI::ComboboxProperties::ValidationString . . . . .	1182
CEGUI::DragContainerProperties::DragAlpha . . . . .	224
CEGUI::DragContainerProperties::DragCursorImage . . . . .	239
CEGUI::DragContainerProperties::DraggingEnabled . . . . .	246
CEGUI::DragContainerProperties::DragThreshold . . . . .	250
CEGUI::EditboxProperties::ActiveSelectionColour . . . . .	86
CEGUI::EditboxProperties::CaratIndex . . . . .	117
CEGUI::EditboxProperties::InactiveSelectionColour . . . . .	472
CEGUI::EditboxProperties::MaskCodepoint . . . . .	589
CEGUI::EditboxProperties::MaskText . . . . .	591
CEGUI::EditboxProperties::MaxTextLength . . . . .	601
CEGUI::EditboxProperties::NormalTextColour . . . . .	711
CEGUI::EditboxProperties::ReadOnly . . . . .	776
CEGUI::EditboxProperties::SelectedTextColour . . . . .	887

CEGUI::EditboxProperties::SelectionLength . . . . .	893
CEGUI::EditboxProperties::SelectionStart . . . . .	897
CEGUI::EditboxProperties::ValidationString . . . . .	1184
CEGUI::FrameWindowProperties::CloseButtonEnabled . . . . .	135
CEGUI::FrameWindowProperties::DragMovingEnabled . . . . .	248
CEGUI::FrameWindowProperties::EWSizingCursorImage . . . . .	285
CEGUI::FrameWindowProperties::FrameEnabled . . . . .	366
CEGUI::FrameWindowProperties::NESWSizingCursorImage . . . . .	705
CEGUI::FrameWindowProperties::NSSizingCursorImage . . . . .	713
CEGUI::FrameWindowProperties::NWSEizingCursorImage . . . . .	717
CEGUI::FrameWindowProperties::RollUpEnabled . . . . .	808
CEGUI::FrameWindowProperties::RollUpState . . . . .	810
CEGUI::FrameWindowProperties::SizingBorderThickness . . . . .	907
CEGUI::FrameWindowProperties::SizingEnabled . . . . .	911
CEGUI::FrameWindowProperties::TitlebarEnabled . . . . .	1110
CEGUI::ItemEntryProperties::Selectable . . . . .	879
CEGUI::ItemEntryProperties::Selected . . . . .	883
CEGUI::ItemListBaseProperties::AutoResizeEnabled . . . . .	100
CEGUI::ItemListBaseProperties::SortEnabled . . . . .	938
CEGUI::ItemListBaseProperties::SortMode . . . . .	942
CEGUI::ItemListboxProperties::MultiSelect . . . . .	699
CEGUI::ListboxProperties::ForceHorzScrollbar . . . . .	344
CEGUI::ListboxProperties::ForceVertScrollbar . . . . .	356
CEGUI::ListboxProperties::ItemTooltips . . . . .	513
CEGUI::ListboxProperties::MultiSelect . . . . .	701
CEGUI::ListboxProperties::Sort . . . . .	926
CEGUI::ListHeaderProperties::ColumnsMovable . . . . .	144
CEGUI::ListHeaderProperties::ColumnsSizable . . . . .	148
CEGUI::ListHeaderProperties::SortColumnID . . . . .	930
CEGUI::ListHeaderProperties::SortDirection . . . . .	934
CEGUI::ListHeaderProperties::SortSettingEnabled . . . . .	946
CEGUI::ListHeaderSegmentProperties::Clickable . . . . .	124
CEGUI::ListHeaderSegmentProperties::Dragable . . . . .	222
CEGUI::ListHeaderSegmentProperties::MovingCursorImage . . . . .	638
CEGUI::ListHeaderSegmentProperties::Sizable . . . . .	904
CEGUI::ListHeaderSegmentProperties::SizingCursorImage . . . . .	909
CEGUI::ListHeaderSegmentProperties::SortDirection . . . . .	932
CEGUI::MenuBaseProperties::AllowMultiplePopups . . . . .	88
CEGUI::MenuBaseProperties::ItemSpacing . . . . .	509
CEGUI::MultiColumnListProperties::ColumnHeader . . . . .	142
CEGUI::MultiColumnListProperties::ColumnsMovable . . . . .	146
CEGUI::MultiColumnListProperties::ColumnsSizable . . . . .	150
CEGUI::MultiColumnListProperties::ForceHorzScrollbar . . . . .	346
CEGUI::MultiColumnListProperties::ForceVertScrollbar . . . . .	360
CEGUI::MultiColumnListProperties::NominatedSelectionColumnID . . . . .	707
CEGUI::MultiColumnListProperties::NominatedSelectionRow . . . . .	709
CEGUI::MultiColumnListProperties::RowCount . . . . .	812
CEGUI::MultiColumnListProperties::SelectionMode . . . . .	895
CEGUI::MultiColumnListProperties::SortColumnID . . . . .	928
CEGUI::MultiColumnListProperties::SortDirection . . . . .	936
CEGUI::MultiColumnListProperties::SortSettingEnabled . . . . .	944
CEGUI::MultiLineEditboxProperties::CaratIndex . . . . .	113
CEGUI::MultiLineEditboxProperties::ForceVertScrollbar . . . . .	348
CEGUI::MultiLineEditboxProperties::MaxTextLength . . . . .	599



CEGUI::MultiLineEditboxProperties::ReadOnly	780
CEGUI::MultiLineEditboxProperties::SelectionBrushImage	889
CEGUI::MultiLineEditboxProperties::SelectionLength	891
CEGUI::MultiLineEditboxProperties::SelectionStart	899
CEGUI::MultiLineEditboxProperties::WordWrap	1326
CEGUI::PopupMenuProperties::FadeInTime	292
CEGUI::PopupMenuProperties::FadeOutTime	294
CEGUI::ProgressBarProperties::CurrentProgress	199
CEGUI::ProgressBarProperties::StepSize	962
CEGUI::PropertyDefinitionBase	751
CEGUI::PropertyDefinition	748
CEGUI::PropertyLinkDefinition	759
CEGUI::RadioButtonProperties::GroupID	403
CEGUI::RadioButtonProperties::Selected	881
CEGUI::ScrollablePaneProperties::ContentArea	185
CEGUI::ScrollablePaneProperties::ContentPaneAutoSized	189
CEGUI::ScrollablePaneProperties::ForceHorzScrollbar	336
CEGUI::ScrollablePaneProperties::ForceVertScrollbar	350
CEGUI::ScrollablePaneProperties::HorzOverlapSize	416
CEGUI::ScrollablePaneProperties::HorzScrollPosition	420
CEGUI::ScrollablePaneProperties::HorzStepSize	422
CEGUI::ScrollablePaneProperties::VertOverlapSize	1192
CEGUI::ScrollablePaneProperties::VertScrollPosition	1196
CEGUI::ScrollablePaneProperties::VertStepSize	1198
CEGUI::ScrollbarProperties::DocumentSize	220
CEGUI::ScrollbarProperties::OverlapSize	721
CEGUI::ScrollbarProperties::PageSize	723
CEGUI::ScrollbarProperties::ScrollPosition	871
CEGUI::ScrollbarProperties::StepSize	966
CEGUI::ScrolledContainerProperties::ChildExtentsArea	122
CEGUI::ScrolledContainerProperties::ContentArea	187
CEGUI::ScrolledContainerProperties::ContentPaneAutoSized	191
CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar	338
CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar	352
CEGUI::SliderProperties::ClickStepSize	126
CEGUI::SliderProperties::CurrentValue	201
CEGUI::SliderProperties::MaximumValue	595
CEGUI::SpinnerProperties::CurrentValue	203
CEGUI::SpinnerProperties::MaximumValue	597
CEGUI::SpinnerProperties::MinimumValue	620
CEGUI::SpinnerProperties::StepSize	964
CEGUI::SpinnerProperties::TextInputMode	1089
CEGUI::TabControlProperties::TabHeight	1076
CEGUI::TabControlProperties::TabPagePosition	1078
CEGUI::TabControlProperties::TabTextPadding	1080
CEGUI::ThumbProperties::HorzFree	414
CEGUI::ThumbProperties::HorzRange	418
CEGUI::ThumbProperties::HotTracked	424
CEGUI::ThumbProperties::VertFree	1188
CEGUI::ThumbProperties::VertRange	1194
CEGUI::TitlebarProperties::DraggingEnabled	244
CEGUI::TooltipProperties::DisplayTime	216
CEGUI::TooltipProperties::FadeTime	296

CEGUI::TooltipProperties::HoverTime . . . . .	426
CEGUI::TreeProperties::ForceHorzScrollbar . . . . .	342
CEGUI::TreeProperties::ForceVertScrollbar . . . . .	358
CEGUI::TreeProperties::ItemTooltips . . . . .	511
CEGUI::TreeProperties::MultiSelect . . . . .	697
CEGUI::TreeProperties::Sort . . . . .	924
CEGUI::WindowProperties::Alpha . . . . .	90
CEGUI::WindowProperties::AlwaysOnTop . . . . .	94
CEGUI::WindowProperties::AutoRepeatDelay . . . . .	96
CEGUI::WindowProperties::AutoRepeatRate . . . . .	98
CEGUI::WindowProperties::ClippedByParent . . . . .	128
CEGUI::WindowProperties::CustomTooltipType . . . . .	205
CEGUI::WindowProperties::DestroyedByParent . . . . .	209
CEGUI::WindowProperties::Disabled . . . . .	214
CEGUI::WindowProperties::DistributeCapturedInputs . . . . .	218
CEGUI::WindowProperties::DragDropTarget . . . . .	242
CEGUI::WindowProperties::Font . . . . .	309
CEGUI::WindowProperties::HorizontalAlignment . . . . .	412
CEGUI::WindowProperties::ID . . . . .	428
CEGUI::WindowProperties::InheritsAlpha . . . . .	474
CEGUI::WindowProperties::InheritsTooltipText . . . . .	476
CEGUI::WindowProperties::LookNFeel . . . . .	587
CEGUI::WindowProperties::MouseButtonDownAutoRepeat . . . . .	622
CEGUI::WindowProperties::MouseCursorImage . . . . .	632
CEGUI::WindowProperties::MousePassThroughEnabled . . . . .	636
CEGUI::WindowProperties::RestoreOldCapture . . . . .	804
CEGUI::WindowProperties::RiseOnClick . . . . .	806
CEGUI::WindowProperties::Text . . . . .	1082
CEGUI::WindowProperties::Tooltip . . . . .	1112
CEGUI::WindowProperties::UnifiedAreaRect . . . . .	1157
CEGUI::WindowProperties::UnifiedHeight . . . . .	1161
CEGUI::WindowProperties::UnifiedMaxSize . . . . .	1163
CEGUI::WindowProperties::UnifiedMinSize . . . . .	1165
CEGUI::WindowProperties::UnifiedPosition . . . . .	1167
CEGUI::WindowProperties::UnifiedSize . . . . .	1169
CEGUI::WindowProperties::UnifiedWidth . . . . .	1171
CEGUI::WindowProperties::UnifiedXPosition . . . . .	1173
CEGUI::WindowProperties::UnifiedYPosition . . . . .	1175
CEGUI::WindowProperties::VerticalAlignment . . . . .	1190
CEGUI::WindowProperties::Visible . . . . .	1200
CEGUI::WindowProperties::WantsMultiClickEvents . . . . .	1202
CEGUI::WindowProperties::WindowRenderer . . . . .	1323
CEGUI::WindowProperties::ZOrderChangeEnabled . . . . .	1339
CEGUI::PropertyHelper . . . . .	756
CEGUI::PropertyInitialiser . . . . .	757
CEGUI::PropertyReceiver . . . . .	762
CEGUI::PropertySet . . . . .	763
CEGUI::Font . . . . .	311
CEGUI::FreeTypeFont . . . . .	387
CEGUI::PixmapFont . . . . .	725
CEGUI::Window . . . . .	1222
CEGUI::RawDataContainer . . . . .	774
CEGUI::Rect . . . . .	782

CEGUI::RefCounted< T > . . . . .	785
CEGUI::Event::ScopedConnection . . . . .	277
CEGUI::RefCounted< CEGUI::BoundSlot > . . . . .	785
CEGUI::RegexValidator . . . . .	787
CEGUI::RenderCache . . . . .	788
CEGUI::ResourceProvider . . . . .	801
CEGUI::Scheme . . . . .	814
CEGUI::Scheme_xmlHandler . . . . .	817
CEGUI::SchemeManager . . . . .	819
CEGUI::ScriptFunctor . . . . .	824
CEGUI::ScriptModule . . . . .	825
CEGUI::ScriptWindowHelper . . . . .	830
CEGUI::SectionSpecification . . . . .	873
CEGUI::SimpleTimer . . . . .	901
CEGUI::Size . . . . .	906
CEGUI::SlotFunctorBase . . . . .	923
CEGUI::FreeFunctionSlot . . . . .	386
CEGUI::FunctorCopySlot< T > . . . . .	391
CEGUI::FunctorPointerSlot< T > . . . . .	392
CEGUI::FunctorReferenceSlot< T > . . . . .	394
CEGUI::MemberFunctionSlot< T > . . . . .	604
CEGUI::StateImagery . . . . .	959
CEGUI::String . . . . .	968
CEGUI::String::const_iterator . . . . .	1035
CEGUI::String::iterator . . . . .	1037
CEGUI::String::FastLessCompare . . . . .	1036
CEGUI::SubComp . . . . .	1038
CEGUI::SubscriberSlot . . . . .	1039
CEGUI::Texture . . . . .	1091
CEGUI::TextUtils . . . . .	1095
CEGUI::TreeItem . . . . .	1144
CEGUI::UDim . . . . .	1156
CEGUI::URect . . . . .	1180
CEGUI::UVector2 . . . . .	1181
CEGUI::Vector2 . . . . .	1186
CEGUI::Vector3 . . . . .	1187
CEGUI::WidgetComponent . . . . .	1204
CEGUI::WidgetLookFeel . . . . .	1208
CEGUI::WidgetLookManager . . . . .	1217
CEGUI::WindowFactory . . . . .	1299
CEGUI::WindowFactoryManager . . . . .	1301
CEGUI::WindowFactoryManager::AliasTargetStack . . . . .	1308
CEGUI::WindowFactoryManager::FalagardWindowMapping . . . . .	1309
CEGUI::WindowManager . . . . .	1310
CEGUI::WindowRenderer . . . . .	1318
CEGUI::EditboxWindowRenderer . . . . .	267
CEGUI::ItemEntryWindowRenderer . . . . .	486
CEGUI::ItemListBaseWindowRenderer . . . . .	501
CEGUI::ListboxWindowRenderer . . . . .	550
CEGUI::ListHeaderWindowRenderer . . . . .	581
CEGUI::MultiColumnListWindowRenderer . . . . .	675
CEGUI::MultiLineEditboxWindowRenderer . . . . .	694
CEGUI::ScrollablePaneWindowRenderer . . . . .	846

CEGUI::ScrollbarWindowRenderer . . . . .	859
CEGUI::SliderWindowRenderer . . . . .	920
CEGUI::TabControlWindowRenderer . . . . .	1073
CEGUI::TooltipWindowRenderer . . . . .	1123
CEGUI::WindowRendererFactory . . . . .	1325
CEGUI::XMLAttributes . . . . .	1328
CEGUI::XMLParser . . . . .	1333
CEGUI::XMLSerializer . . . . .	1336

## Chapter 3

# Crazy Eddies GUI System Class Index

### 3.1 Crazy Eddies GUI System Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CEGUI::AbsoluteDim (Dimension type that represents an absolute pixel value. Implements BaseDim interface ) . . . . .	83
CEGUI::ActivationEventArgs (EventArgs based class that is used for Activated and Deactivated window events ) . . . . .	85
CEGUI::EditboxProperties::ActiveSelectionColour (Property to access the colour used for rendering the selection highlight when the edit box is active ) . . . . .	86
CEGUI::MenuBaseProperties::AllowMultiplePopups (Property to access the state of the allow multiple popups setting ) . . . . .	88
CEGUI::WindowProperties::Alpha (Property to access window alpha setting ) . . . . .	90
CEGUI::AlreadyExistsException (Exception class used when an attempt is made to use an object name that is already in use within the system ) . . . . .	92
CEGUI::WindowProperties::AlwaysOnTop (Property to access window "Always-On-Top" setting ) . . . . .	94
CEGUI::WindowProperties::AutoRepeatDelay (Property to access window autorepeat delay value ) . . . . .	96
CEGUI::WindowProperties::AutoRepeatRate (Property to access window autorepeat rate value ) . . . . .	98
CEGUI::ItemListBaseProperties::AutoResizeEnabled (Property to access the state of the auto resize enabled setting ) . . . . .	100
CEGUI::BaseDim (Abstract interface for a generic 'dimension' class ) . . . . .	102
CEGUI::BoundSlot (Class that tracks a SubscriberSlot, its group, and the Event to which it was subscribed. This is effectively what gets returned from the calls to the Event::subscribe members, though BoundSlot is always wrapped in a reference counted pointer. When a BoundSlot is deleted, the connection is unsubscribed and the SubscriberSlot is deleted ) . . . . .	106
CEGUI::ButtonBase (Base class for all the 'button' type widgets (push button, radio button, check-box, etc) ) . . . . .	109
CEGUI::MultiLineEditboxProperties::CaratIndex (Property to access the current carat index ) . . . . .	113
CEGUI::ComboboxProperties::CaratIndex (Property to access the current carat index ) . . . . .	115
CEGUI::EditboxProperties::CaratIndex (Property to access the current carat index ) . . . . .	117
CEGUI::Checkbox (Base class providing logic for Check-box widgets ) . . . . .	119
CEGUI::ScrolledContainerProperties::ChildExtentsArea (Property offering read-only access to the current content extents rectangle (as window relative pixels) ) . . . . .	122
CEGUI::ListHeaderSegmentProperties::Clickable (Property to access the click-able setting of the header segment ) . . . . .	124

CEGUI::SliderProperties::ClickStepSize (Property to access the click-step size for the slider) . .	126
CEGUI::WindowProperties::ClippedByParent (Property to access window "clipped by parent" setting) . . . . .	128
CEGUI::ClippedContainer (Helper container window that has configurable clipping. Used by the <code>ItemListbox</code> widget) . . . . .	130
CEGUI::FrameWindowProperties::CloseButtonEnabled (Property to access the setting for whether the window close button will be enabled (or displayed depending upon choice of final widget type)) . . . . .	135
CEGUI::colour (Class representing colour values within the system) . . . . .	137
CEGUI::ColourRect (Class that holds details of colours for the four corners of a rectangle) . .	138
CEGUI::MultiColumnListProperties::ColumnHeader (Property to access a column) . . . . .	142
CEGUI::ListHeaderProperties::ColumnsMovable (Property to access the setting for user moving of the column headers) . . . . .	144
CEGUI::MultiColumnListProperties::ColumnsMovable (Property to access the setting for user moving of the column headers) . . . . .	146
CEGUI::ListHeaderProperties::ColumnsSizable (Property to access the setting for user sizing of the column headers) . . . . .	148
CEGUI::MultiColumnListProperties::ColumnsSizable (Property to access the setting for user sizing of the column headers) . . . . .	150
CEGUI::Combobox (Base class for the <code>Combobox</code> widget) . . . . .	152
CEGUI::ComboDropList (Base class for the combo box drop down list. This is a specialisation of the <code>Listbox</code> class) . . . . .	171
CEGUI::ComponentArea (Class that represents a target area for a widget or imagery component) .	177
CEGUI::Config_xmlHandler (Handler class used to parse the Configuration XML file) . . . . .	180
CEGUI::ConstBaseIterator< T > (Base class constant iterator used to offer iteration over various collections within the system) . . . . .	182
CEGUI::ScrollablePaneProperties::ContentArea (Property to access the current content pane area rectangle (as window relative pixels)) . . . . .	185
CEGUI::ScrolledContainerProperties::ContentArea (Property to access the current content pane area rectangle (as window relative pixels)) . . . . .	187
CEGUI::ScrollablePaneProperties::ContentPaneAutoSize (Property to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content) . . . . .	189
CEGUI::ScrolledContainerProperties::ContentPaneAutoSize (Property to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content) . . . . .	191
CEGUI::CoordConverter (Utility class that helps in converting various types of co-ordinate between absolute screen positions and positions offset from the top-left corner of a given <code>Window</code> object) . . . . .	193
CEGUI::ProgressBarProperties::CurrentProgress (Property to access the current progress of the progress bar) . . . . .	199
CEGUI::SliderProperties::CurrentValue (Property to access the current value of the slider) . . .	201
CEGUI::SpinnerProperties::CurrentValue (Property to access the current value of the spinner) .	203
CEGUI::WindowProperties::CustomTooltipType (Property to access the custom tooltip for this <code>Window</code> ) . . . . .	205
CEGUI::DefaultLogger (Default implementation for the <code>Logger</code> class. If you want to redirect CEGUI logs to some place other than a text file, implement your own <code>Logger</code> implementation and create a object of the <code>Logger</code> type before creating the <code>CEGUI::System</code> singleton) . . . . .	207
CEGUI::WindowProperties::DestroyedByParent (Property to access window Destroyed by Parent setting) . . . . .	209
CEGUI::Dimension (Class representing some kind of dimension) . . . . .	211
CEGUI::WindowProperties::Disabled (Property to access window <code>Disabled</code> setting) . . . . .	214

CEGUI::TooltipProperties::DisplayTime (Property to access the time after which the tooltip automatically de-activates itself ) . . . . .	216
CEGUI::WindowProperties::DistributeCapturedInputs (Property to access whether inputs are passed to child windows when input is captured to this window ) . . . . .	218
CEGUI::ScrollbarProperties::DocumentSize (Property to access the document size for the Scrollbar ) . . . . .	220
CEGUI::ListHeaderSegmentProperties::Dragable (Property to access the drag-able setting of the header segment ) . . . . .	222
CEGUI::DragContainerProperties::DragAlpha (Property to access the dragging alpha value ) . .	224
CEGUI::DragContainer (Generic drag & drop enabled window class ) . . . . .	226
CEGUI::DragContainerProperties::DragCursorImage (Property to access the dragging mouse cursor setting ) . . . . .	239
CEGUI::DragDropEventArgs (EventArgs based class used for certain drag/drop notifications ) .	241
CEGUI::WindowProperties::DragDropTarget (Property to get/set whether the Window will receive drag and drop related notifications ) . . . . .	242
CEGUI::TitlebarProperties::DraggingEnabled (Property to access the state of the dragging enabled setting for the Titlebar ) . . . . .	244
CEGUI::DragContainerProperties::DraggingEnabled (Property to access the state of the dragging enabled setting ) . . . . .	246
CEGUI::FrameWindowProperties::DragMovingEnabled (Property to access the setting for whether the user may drag the window around by its title bar ) . . . . .	248
CEGUI::DragContainerProperties::DragThreshold (Property to access the dragging threshold value ) . . . . .	250
CEGUI::DynamicModule (Class that wraps and gives access to a dynamically linked module (.dll, .so, etc...) ) . . . . .	252
CEGUI::Editbox (Base class for an Editbox widget ) . . . . .	254
CEGUI::EditboxWindowRenderer (Base class for the EditboxWindowRenderer class ) . . . . .	267
CEGUI::ComboboxProperties::EditSelectionLength (Property to access the current selection length ) . . . . .	270
CEGUI::ComboboxProperties::EditSelectionStart (Property to access the current selection start index ) . . . . .	272
CEGUI::Event (Defines an 'event' which can be subscribed to by interested parties ) . . . . .	274
CEGUI::Event::ScopedConnection (Event::Connection wrapper that automatically disconnects the connection when the object is deleted (or goes out of scope) ) . . . . .	277
CEGUI::EventArgs (Base class used as the argument to all subscribers Event object ) . . . . .	278
CEGUI::EventSet (Class that collects together a set of Event objects ) . . . . .	279
CEGUI::FrameWindowProperties::EWSizingCursorImage (Property to access the E-W (left/right) sizing cursor image ) . . . . .	285
CEGUI::Exception (Root exception class used within the GUI system ) . . . . .	287
CEGUI::FactoryModule (Class that encapsulates access to a dynamic loadable module containing implementations of Windows, Widgets, and their factories ) . . . . .	290
CEGUI::PopupMenuProperties::FadeInTime (Property to access the fade in time in seconds of the popup menu ) . . . . .	292
CEGUI::PopupMenuProperties::FadeOutTime (Property to access the fade out time in seconds of the popup menu ) . . . . .	294
CEGUI::TooltipProperties::FadeTime (Property to access the duration of the fade effect for the tooltip ) . . . . .	296
CEGUI::Falagard_xmlHandler (Handler class used to parse look & feel XML files used by the Falagard system ) . . . . .	298
CEGUI::FalagardComponentBase (Common base class used for renderable components within an ImagerySection ) . . . . .	299
CEGUI::FalagardXMLHelper (Utility helper class primarily intended for use by the falagard xml parser ) . . . . .	306
CEGUI::FileIOException (Exception class used when a file handling problem occurs ) . . . . .	307



CEGUI::WindowProperties::Font (Property to access window Font setting ) . . . . .	309
CEGUI::Font (Class that encapsulates text rendering functionality for a typeface ) . . . . .	311
CEGUI::Font_xmlHandler (Handler class used to parse the Font XML files using SAX2 ) . . . . .	326
CEGUI::FontDim (Dimension type that represents some metric of a Font. Implements BaseDim interface ) . . . . .	327
CEGUI::FontGlyph (Internal class representing a single font glyph ) . . . . .	329
CEGUI::FontManager (Class providing a shared library of Font objects to the system ) . . . . .	331
CEGUI::ScrollablePaneProperties::ForceHorzScrollbar (Property to access the setting which controls whether the horizontal scroll bar will always be displayed, or only displayed when it is required ) . . . . .	336
CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar (Property to access the state of the force horizontal scrollbar setting ) . . . . .	338
CEGUI::ComboboxProperties::ForceHorzScrollbar (Property to access the 'always show' setting for the horizontal scroll bar of the list box ) . . . . .	340
CEGUI::TreeProperties::ForceHorzScrollbar (Property to access the 'always show' setting for the horizontal scroll bar of the list box ) . . . . .	342
CEGUI::ListboxProperties::ForceHorzScrollbar (Property to access the 'always show' setting for the horizontal scroll bar of the list box ) . . . . .	344
CEGUI::MultiColumnListProperties::ForceHorzScrollbar (Property to access the 'always show' setting for the horizontal scroll bar of the list box ) . . . . .	346
CEGUI::MultiLineEditboxProperties::ForceVertScrollbar (Property to access the 'always show' setting for the vertical scroll bar of the box ) . . . . .	348
CEGUI::ScrollablePaneProperties::ForceVertScrollbar (Property to access the setting which controls whether the vertical scroll bar will always be displayed, or only displayed when it is required ) . . . . .	350
CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar (Property to access the state of the force vertical scrollbar setting ) . . . . .	352
CEGUI::ComboboxProperties::ForceVertScrollbar (Property to access the 'always show' setting for the vertical scroll bar of the list box ) . . . . .	354
CEGUI::ListboxProperties::ForceVertScrollbar (Property to access the 'always show' setting for the vertical scroll bar of the list box ) . . . . .	356
CEGUI::TreeProperties::ForceVertScrollbar (Property to access the 'always show' setting for the vertical scroll bar of the list box ) . . . . .	358
CEGUI::MultiColumnListProperties::ForceVertScrollbar (Property to access the 'always show' setting for the vertical scroll bar of the list box ) . . . . .	360
CEGUI::FrameComponent (Class that encapsulates information for a frame with background (9 images in total) ) . . . . .	362
CEGUI::FrameWindowProperties::FrameEnabled (Property to access the setting for whether the window frame will be displayed ) . . . . .	366
CEGUI::FrameWindow (Abstract base class for a movable, sizable, window with a title-bar and a frame ) . . . . .	368
CEGUI::FreeFunctionSlot (Slot functor class that calls back via a free function pointer ) . . . . .	386
CEGUI::FreeTypeFont (Implementation of the Font class interface using the FreeType library ) . . . . .	387
CEGUI::FunctorCopySlot< T > (Slot template class that creates a functor that calls back via a copy of a functor object ) . . . . .	391
CEGUI::FunctorPointerSlot< T > (Slot template class that creates a functor that calls back via a functor object pointer ) . . . . .	392
CEGUI::FunctorReferenceBinder< T > (Class that enables the creation of a reference binding for a functor object to be used as a callback slot. Wrap your functor with this to enable the use of an object reference when subscribing to an event signal (as opposed to the functor object being copied, or using a pointer) ) . . . . .	393
CEGUI::FunctorReferenceSlot< T > (Slot template class that creates a functor that calls back via a functor object reference ) . . . . .	394
CEGUI::GenericException (Exception class used when none of the other classes are applicable ) . . . . .	395



CEGUI::GlobalEventSet (The <a href="#">GlobalEventSet</a> singleton allows you to subscribe to an event for all instances of a class. The <a href="#">GlobalEventSet</a> effectively supports "late binding" to events; which means you can subscribe to some event that does not actually exist (yet) )	397
CEGUI::GroupBox (Base class for standard <a href="#">GroupBox</a> widget )	399
CEGUI::RadioButtonProperties::GroupID (Property to access the radio button group ID )	403
CEGUI::GUILayout_xmlHandler (Handler class used to parse the GUILayout XML files using SAX2 )	405
CEGUI::GUISheet (Window class intended to be used as a simple, generic <a href="#">Window</a> )	408
CEGUI::HeaderSequenceEventArgs (EventArgs class used for segment move (sequence changed) events )	411
CEGUI::WindowProperties::HorizontalAlignment (Property to access the horizontal alignment setting for the window )	412
CEGUI::ThumbProperties::HorzFree (Property to access the state the setting to free the thumb horizontally )	414
CEGUI::ScrollablePaneProperties::HorzOverlapSize (Property to access the overlap size for the horizontal <a href="#">Scrollbar</a> )	416
CEGUI::ThumbProperties::HorzRange (Property to access the horizontal movement range for the thumb )	418
CEGUI::ScrollablePaneProperties::HorzScrollPosition (Property to access the scroll position of the horizontal <a href="#">Scrollbar</a> )	420
CEGUI::ScrollablePaneProperties::HorzStepSize (Property to access the step size for the horizontal <a href="#">Scrollbar</a> )	422
CEGUI::ThumbProperties::HotTracked (Property to access the state of the "hot-tracked" setting for the thumb )	424
CEGUI::TooltipProperties::HoverTime (Property to access the timeout that must expire before the tooltip gets activated )	426
CEGUI::WindowProperties::ID (Property to access window ID field )	428
CEGUI::Image (Class that represents a single <a href="#">Image</a> of an <a href="#">Imageset</a> )	430
CEGUI::ImageCodec (Abstract ImageLoader class. An image loader encapsulate the loading of a texture )	439
CEGUI::ImageDim (Dimension type that represents some dimension of a named <a href="#">Image</a> . Implements <a href="#">BaseDim</a> interface )	441
CEGUI::ImageryComponent (Class that encapsulates information for a single image component )	443
CEGUI::ImagerySection (Class that encapsulates a re-usable collection of imagery specifications )	448
CEGUI::Imageset (Offers functions to define, access, and draw, a set of image components on a single graphical surface or <a href="#">Texture</a> )	454
CEGUI::Imageset_xmlHandler (Handler class used to parse the <a href="#">Imageset</a> XML files using SAX2 )	466
CEGUI::ImagesetManager (Class providing a shared library of <a href="#">Imageset</a> objects to the system )	468
CEGUI::EditboxProperties::InactiveSelectionColour (Property to access the colour used for rendering the selection highlight when the edit box is inactive )	472
CEGUI::WindowProperties::InheritsAlpha (Property to access window "Inherits Alpha" setting )	474
CEGUI::WindowProperties::InheritsTooltipText (Property to access whether the window inherits its tooltip text from its parent whn it has no tooltip text of its own )	476
CEGUI::InvalidRequestException (Exception class used when some impossible request was made for the current system state )	478
CEGUI::ItemEntry (Base class for item type widgets )	480
CEGUI::ItemEntryWindowRenderer (Base class for <a href="#">ItemEntry</a> window renderer objects )	486
CEGUI::ItemListBase (Base class for item list widgets )	489
CEGUI::ItemListBaseWindowRenderer (Base class for <a href="#">ItemListBase</a> window renderer )	501
CEGUI::ItemListbox (ItemListbox window class )	504
CEGUI::MenuBaseProperties::ItemSpacing (Property to access the item spacing of the menu )	509
CEGUI::TreeProperties::ItemTooltips (Property to access the show item tooltips setting of the list box )	511

CEGUI::ListboxProperties::ItemTooltips (Property to access the show item tooltips setting of the list box ) . . . . .	513
CEGUI::Key (Struct to give scope to scan code enumeration ) . . . . .	515
CEGUI::KeyEventArgs (EventArgs based class that is used for objects passed to input event handlers concerning keyboard input ) . . . . .	517
CEGUI::LayerSpecification (Class that encapsulates a single layer of imagery ) . . . . .	518
CEGUI::Listbox (Base class for standard Listbox widget ) . . . . .	521
CEGUI::ListboxItem (Base class for list box items ) . . . . .	537
CEGUI::ListboxTextItem (Class used for textual items in a list box ) . . . . .	546
CEGUI::ListboxWindowRenderer (Base class for Listbox window renderer ) . . . . .	550
CEGUI::ListHeader (Base class for the multi column list header widget ) . . . . .	553
CEGUI::ListHeaderSegment (Base class for list header segment window ) . . . . .	571
CEGUI::ListHeaderWindowRenderer (Base class for the multi column list header window renderer ) . . . . .	581
CEGUI::Logger (Abstract class that defines the interface of a logger object for the GUI system. The default implementation of this interface is the DefaultLogger class; if you want to perform special logging, derive your own class from Logger and initialize a object of that type before you create the CEGUI::System singleton ) . . . . .	584
CEGUI::WindowProperties::LookNFeel (Property to access/change the assigned look'n'feel ) . . . . .	587
CEGUI::EditboxProperties::MaskCodepoint (Property to access the mask text setting of the edit box ) . . . . .	589
CEGUI::EditboxProperties::MaskText (Property to access the mask text setting of the edit box ) . . . . .	591
CEGUI::ComboboxProperties::MaxEditTextLength (Property to access the maximum text length for the edit box ) . . . . .	593
CEGUI::SliderProperties::MaximumValue (Property to access the maximum value of the slider ) . . . . .	595
CEGUI::SpinnerProperties::MaximumValue (Property to access the maximum value setting of the spinner ) . . . . .	597
CEGUI::MultiLineEditboxProperties::MaxTextLength (Property to access the maximum text length for the edit box ) . . . . .	599
CEGUI::EditboxProperties::MaxTextLength (Property to access the maximum text length for the edit box ) . . . . .	601
CEGUI::MCLGridRef (Simple grid index structure ) . . . . .	603
CEGUI::MemberFunctionSlot< T > (Slot template class that creates a functor that calls back via a class member function ) . . . . .	604
CEGUI::MemoryException (Exception class used when a memory handling error is detected ) . . . . .	605
CEGUI::Menubar (Base class for menu bars ) . . . . .	607
CEGUI::MenuBase (Abstract base class for menus ) . . . . .	610
CEGUI::MenuItem (Base class for menu items ) . . . . .	613
CEGUI::SpinnerProperties::MinimumValue (Property to access the minimum value setting of the spinner ) . . . . .	620
CEGUI::WindowProperties::MouseButtonDownAutoRepeat (Property to control whether the window will receive autorepeat mouse button down events ) . . . . .	622
CEGUI::MouseClickedTracker (Implementation structure used in tracking up & down mouse button inputs in order to generate click, double-click, and triple-click events ) . . . . .	624
CEGUI::MouseCursor (Class that allows access to the GUI system mouse cursor ) . . . . .	625
CEGUI::MouseCursorEventArgs (EventArgs based class that is used for objects passed to input event handlers concerning mouse cursor events ) . . . . .	631
CEGUI::WindowProperties::MouseCursorImage (Property to access window mouse cursor setting ) . . . . .	632
CEGUI::MouseEventArgs (EventArgs based class that is used for objects passed to input event handlers concerning mouse input ) . . . . .	634
CEGUI::WindowProperties::MousePassThroughEnabled (Property to access whether the window ignores mouse events and pass them through to any windows behind it ) . . . . .	636

CEGUI::ListHeaderSegmentProperties::MovingCursorImage (Property to access the segment moving cursor image ) . . . . .	638
CEGUI::MultiColumnList (Base class for the multi column list widget ) . . . . .	640
CEGUI::MultiColumnList::ListRow (Struct used internally to represent a row in the list and also to ease sorting of the rows ) . . . . .	674
CEGUI::MultiColumnListWindowRenderer (Base class for the multi column list window renderer ) . . . . .	675
CEGUI::MultiLineEditbox (Base class for the multi-line edit box widget ) . . . . .	678
CEGUI::MultiLineEditbox::LineInfo (Struct used to store information about a formatted line within the paragraph ) . . . . .	693
CEGUI::MultiLineEditboxWindowRenderer (Base class for multi-line edit box window renderer objects ) . . . . .	694
CEGUI::TreeProperties::MultiSelect (Property to access the multi-select setting of the list box ) . . . . .	697
CEGUI::ItemListboxProperties::MultiSelect (Property to access the state of the multiselect enabled setting ) . . . . .	699
CEGUI::ListboxProperties::MultiSelect (Property to access the multi-select setting of the list box ) . . . . .	701
CEGUI::NamedArea (NamedArea defines an area for a component which may later be obtained and referenced by a name unique to the WidgetLook holding the NamedArea ) . . . . .	703
CEGUI::FrameWindowProperties::NESWSizingCursorImage (Property to access the NE-SW diagonal sizing cursor image ) . . . . .	705
CEGUI::MultiColumnListProperties::NominatedSelectionColumnID (Property to access the nominated selection column (via ID) ) . . . . .	707
CEGUI::MultiColumnListProperties::NominatedSelectionRow (Property to access the nominated selection row ) . . . . .	709
CEGUI::EditboxProperties::NormalTextColour (Property to access the normal, unselected, text colour used for rendering text ) . . . . .	711
CEGUI::FrameWindowProperties::NSSizingCursorImage (Property to access the N-S (up-down) sizing cursor image ) . . . . .	713
CEGUI::NullObjectException (Exception class used when some required object or parameter is null ) . . . . .	715
CEGUI::FrameWindowProperties::NWSESizingCursorImage (Property to access the NW-SE diagonal sizing cursor image ) . . . . .	717
CEGUI::ObjectInUseException (Exception class used when some attempt to delete, remove, or otherwise invalidate some object that is still in use occurs ) . . . . .	719
CEGUI::ScrollbarProperties::OverlapSize (Property to access the overlap size for the Scrollbar ) . . . . .	721
CEGUI::ScrollbarProperties::PageSize (Property to access the page size for the Scrollbar ) . . . . .	723
CEGUI::PixmapFont (Implementation of the Font class interface using static Imageset's ) . . . . .	725
CEGUI::PopupMenu (Base class for popup menus ) . . . . .	728
CEGUI::ProgressBar (Base class for progress bars ) . . . . .	734
CEGUI::Property (An abstract class that defines the interface to access object properties by name ) . . . . .	739
CEGUI::PropertyDefinition (Class representing a generic get/set property ) . . . . .	748
CEGUI::PropertyDefinitionBase (Common base class used for types representing a new property to be available on all widgets that use the WidgetLook that the property definition is a part of ) . . . . .	751
CEGUI::PropertyDim (Dimension type that represents the value of a Window property. Implements BaseDim interface ) . . . . .	754
CEGUI::PropertyHelper (Helper class used to convert various data types to and from the format expected in Property strings ) . . . . .	756
CEGUI::PropertyInitialiser (Class that holds information about a property and it's required initial value ) . . . . .	757
CEGUI::PropertyLinkDefinition (Class representing a property that links to another property defined on an attached child widget ) . . . . .	759
CEGUI::PropertyReceiver (Dummy base class to ensure correct casting of receivers ) . . . . .	762
CEGUI::PropertySet (Class that contains a collection of Property objects ) . . . . .	763

CEGUI::PushButton (Base class to provide logic for push button type widgets ) . . . . .	767
CEGUI::RadioButton (Base class to provide the logic for Radio Button widgets ) . . . . .	770
CEGUI::RawDataContainer (Class used as the databuffer for loading files throughout the library ) . . . . .	774
CEGUI::EditboxProperties::ReadOnly (Property to access the read-only setting of the edit box ) . . . . .	776
CEGUI::ComboboxProperties::ReadOnly (Property to access the read-only setting of the edit box ) . . . . .	778
CEGUI::MultiLineEditboxProperties::ReadOnly (Property to access the read-only setting of the edit box ) . . . . .	780
CEGUI::Rect (Class encapsulating operations on a Rectangle ) . . . . .	782
CEGUI::RefCounted< T > (Simple, generic, reference counted pointer class. This is primarily here for use by the Events system to track when to delete slot bindings ) . . . . .	785
CEGUI::RegexValidator (Internal struct to contain compiled regex string ) . . . . .	787
CEGUI::RenderCache (Class that acts as a cache for images that need to be rendered ) . . . . .	788
CEGUI::Renderer (Abstract class defining the interface for <a href="#">Renderer</a> objects ) . . . . .	791
CEGUI::RendererException (Exception class used when an problem is detected within the <a href="#">Renderer</a> or related objects ) . . . . .	799
CEGUI::ResourceProvider (Abstract class that defines the required interface for all resource provider sub-classes ) . . . . .	801
CEGUI::WindowProperties::RestoreOldCapture (Property to access window Restore Old Capture setting ) . . . . .	804
CEGUI::WindowProperties::RiseOnClick (Property to access whether the window rises to the top of the z order when clicked ) . . . . .	806
CEGUI::FrameWindowProperties::RollUpEnabled (Property to access the setting for whether the user is able to roll-up / shade the window ) . . . . .	808
CEGUI::FrameWindowProperties::RollUpState (Property to access the roll-up / shade state of the window ) . . . . .	810
CEGUI::MultiColumnListProperties::RowCount (Property to access the number of rows in the list (read-only) ) . . . . .	812
CEGUI::Scheme (A class that groups a set of GUI elements and initialises the system to access those elements ) . . . . .	814
CEGUI::Scheme_xmlHandler (Handler class used to parse the <a href="#">Scheme</a> XML files using SAX2 ) . . . . .	817
CEGUI::SchemeManager (A class that manages the creation of, access to, and destruction of GUI <a href="#">Scheme</a> objects ) . . . . .	819
CEGUI::ScriptException (Exception class used when a scripting error occurs ) . . . . .	822
CEGUI::ScriptFunctor (Functor class used for binding named script functions to events ) . . . . .	824
CEGUI::ScriptModule (Abstract interface required for all scripting support modules to be used with the <a href="#">CEGUI</a> system ) . . . . .	825
CEGUI::ScriptWindowHelper . . . . .	830
CEGUI::ScrollablePane (Base class for the <a href="#">ScrollablePane</a> widget ) . . . . .	832
CEGUI::ScrollablePaneWindowRenderer (Base class for <a href="#">ScrollablePane</a> window renderer objects ) . . . . .	846
CEGUI::Scrollbar (Base scroll bar class ) . . . . .	849
CEGUI::ScrollbarWindowRenderer (Base class for <a href="#">ItemEntry</a> window renderer objects ) . . . . .	859
CEGUI::ScrolledContainer (Helper container window class which is used in the implementation of the <a href="#">ScrollablePane</a> widget class ) . . . . .	862
CEGUI::ScrolledItemListBase ( <a href="#">ScrolledItemListBase</a> window class ) . . . . .	868
CEGUI::ScrollbarProperties::ScrollPosition (Property to access the scroll position of the <a href="#">Scrollbar</a> ) . . . . .	871
CEGUI::SectionSpecification (Class that represents a simple 'link' to an <a href="#">ImagerySection</a> ) . . . . .	873
CEGUI::ItemEntryProperties::Selectable (Property to access the state of the selectable setting ) . . . . .	879
CEGUI::RadioButtonProperties::Selected (Property to access the selected state of the <a href="#">RadioButton</a> ) . . . . .	881
CEGUI::ItemEntryProperties::Selected (Property to access the state of the selected setting ) . . . . .	883
CEGUI::CheckboxProperties::Selected (Property to access the selected state of the check box ) . . . . .	885

CEGUI::EditboxProperties::SelectedTextColour (Property to access the colour used for rendering text within the selection area ) . . . . .	887
CEGUI::MultiLineEditboxProperties::SelectionBrushImage (Property to access the selection brush image ) . . . . .	889
CEGUI::MultiLineEditboxProperties::SelectionLength (Property to access the current selection length ) . . . . .	891
CEGUI::EditboxProperties::SelectionLength (Property to access the current selection length ) . .	893
CEGUI::MultiColumnListProperties::SelectionMode (Property to access the selection mode setting of the list ) . . . . .	895
CEGUI::EditboxProperties::SelectionStart (Property to access the current selection start index ) .	897
CEGUI::MultiLineEditboxProperties::SelectionStart (Property to access the current selection start index ) . . . . .	899
CEGUI::SimpleTimer (Simple timer class ) . . . . .	901
CEGUI::ComboboxProperties::SingleClickMode (Property to access the 'single click mode' setting for the combo box ) . . . . .	902
CEGUI::ListHeaderSegmentProperties::Sizable (Property to access the sizable setting of the header segment ) . . . . .	904
CEGUI::Size (Class that holds the size (width & height) of something ) . . . . .	906
CEGUI::FrameWindowProperties::SizingBorderThickness (Property to access the setting for the sizing border thickness ) . . . . .	907
CEGUI::ListHeaderSegmentProperties::SizingCursorImage (Property to access the segment sizing cursor image ) . . . . .	909
CEGUI::FrameWindowProperties::SizingEnabled (Property to access the state of the sizable setting for the FrameWindow ) . . . . .	911
CEGUI::Slider (Base class for Slider widgets ) . . . . .	913
CEGUI::SliderWindowRenderer (Base class for ItemEntry window renderer objects ) . . . . .	920
CEGUI::SlotFunctorBase (Defines abstract interface which will be used when constructing various functor objects that bind slots to signals (or in CEGUI terms, handlers to events) ) . . . . .	923
CEGUI::TreeProperties::Sort (Property to access the sort setting of the list box ) . . . . .	924
CEGUI::ListboxProperties::Sort (Property to access the sort setting of the list box ) . . . . .	926
CEGUI::MultiColumnListProperties::SortColumnID (Property to access the current sort column (via ID code) ) . . . . .	928
CEGUI::ListHeaderProperties::SortColumnID (Property to access the current sort column (via ID code) ) . . . . .	930
CEGUI::ListHeaderSegmentProperties::SortDirection (Property to access the sort direction setting of the header segment ) . . . . .	932
CEGUI::ListHeaderProperties::SortDirection (Property to access the sort direction setting of the list header ) . . . . .	934
CEGUI::MultiColumnListProperties::SortDirection (Property to access the sort direction setting of the list ) . . . . .	936
CEGUI::ItemListBaseProperties::SortEnabled (Property to access the state of the sorting enabled setting ) . . . . .	938
CEGUI::ComboboxProperties::SortList (Property to access the sort setting of the list box ) . . .	940
CEGUI::ItemListBaseProperties::SortMode (Property to access the sorting mode ) . . . . .	942
CEGUI::MultiColumnListProperties::SortSettingEnabled (Property to access the setting for user modification of the sort column & direction ) . . . . .	944
CEGUI::ListHeaderProperties::SortSettingEnabled (Property to access the setting for user modification of the sort column & direction ) . . . . .	946
CEGUI::Spinner (Base class for the Spinner widget ) . . . . .	948
CEGUI::StateImagery (Class the encapsulates imagery for a given widget state ) . . . . .	959
CEGUI::ProgressBarProperties::StepSize (Property to access the step size setting for the progress bar ) . . . . .	962
CEGUI::SpinnerProperties::StepSize (Property to access the step size of the spinner ) . . . . .	964



CEGUI::ScrollbarProperties::StepSize (Property to access the step size for the <a href="#">Scrollbar</a> ) . . . .	966
CEGUI::String (String class used within the GUI system ) . . . . .	968
CEGUI::String::const_iterator (Constant forward <a href="#">iterator</a> class for <a href="#">String</a> objects ) . . . . .	1035
CEGUI::String::FastLessCompare (Functor that can be used as comparator in a <code>std::map</code> with <a href="#">String</a> keys. It's faster than using the default, but the map will no longer be sorted alphabetically ) . . . . .	1036
CEGUI::String::iterator (Forward <a href="#">iterator</a> class for <a href="#">String</a> objects ) . . . . .	1037
CEGUI::SubComp (Implementation helper functor which is used to locate a <a href="#">BoundSlot</a> in the <code>multimap</code> collection of <a href="#">BoundSlots</a> ) . . . . .	1038
CEGUI::SubscriberSlot ( <a href="#">SubscriberSlot</a> class which is used when subscribing to events ) . . . .	1039
CEGUI::System (The <a href="#">System</a> class is the <a href="#">CEGUI</a> class that provides access to all other elements in this system ) . . . . .	1041
CEGUI::TabButton (Base class for <a href="#">TabButtons</a> . A <a href="#">TabButton</a> based class is used internally as the button that appears at the top of a <a href="#">TabControl</a> widget to select the active tab pane ) . . .	1058
CEGUI::TabControl (Base class for standard <a href="#">Tab Control</a> widget ) . . . . .	1062
CEGUI::TabControlWindowRenderer (Base class for <a href="#">TabControl</a> window renderer objects ) . .	1073
CEGUI::TabControlProperties::TabHeight (Property to access the tab height setting of the tab control ) . . . . .	1076
CEGUI::TabControlProperties::TabPanePosition (Property to query/set the position of the button pane in tab control ) . . . . .	1078
CEGUI::TabControlProperties::TabTextPadding (Property to access the tab text padding setting of the tab control ) . . . . .	1080
CEGUI::WindowProperties::Text (Property to access window text setting ) . . . . .	1082
CEGUI::TextComponent (Class that encapsulates information for a text component ) . . . . .	1084
CEGUI::SpinnerProperties::TextInputMode (Property to access the <a href="#">TextInputMode</a> setting ) . .	1089
CEGUI::Texture (Abstract base class specifying the required interface for <a href="#">Texture</a> objects ) . . .	1091
CEGUI::TextUtils (Text utility support class. This class is all static members. You do not create instances of this class ) . . . . .	1095
CEGUI::Thumb (Base class for <a href="#">Thumb</a> widget ) . . . . .	1098
CEGUI::Titlebar (Class representing the title bar for <a href="#">Frame Windows</a> ) . . . . .	1104
CEGUI::FrameWindowProperties::TitlebarEnabled (Property to access the setting for whether the window title-bar will be enabled (or displayed depending upon choice of final widget type) ) . . . . .	1110
CEGUI::WindowProperties::Tooltip (Property to access the tooltip text for this <a href="#">Window</a> ) . . . .	1112
CEGUI::Tooltip (Base class for <a href="#">Tooltip</a> widgets ) . . . . .	1114
CEGUI::TooltipWindowRenderer (Base class for <a href="#">Tooltip</a> window renderer objects ) . . . . .	1123
CEGUI::Tree (Base class for standard <a href="#">Tree</a> widget ) . . . . .	1126
CEGUI::TreeEventArgs ( <a href="#">EventArgs</a> based class that is used for objects passed to input event handlers concerning <a href="#">Tree</a> events ) . . . . .	1143
CEGUI::TreeItem (Base class for list box items ) . . . . .	1144
CEGUI::UDim (Class representing a unified dimension; that is a dimension that has both a relative 'scale' portion and absolute 'offset' portion ) . . . . .	1156
CEGUI::WindowProperties::UnifiedAreaRect (Property to access the unified area rectangle of the window ) . . . . .	1157
CEGUI::UnifiedDim ( <a href="#">Dimension</a> type that represents an Unified dimension. Implements <a href="#">BaseDim</a> interface ) . . . . .	1159
CEGUI::WindowProperties::UnifiedHeight (Property to access the unified height of the window )	1161
CEGUI::WindowProperties::UnifiedMaxSize (Property to access the unified maximum size of the window ) . . . . .	1163
CEGUI::WindowProperties::UnifiedMinSize (Property to access the unified minimum size of the window ) . . . . .	1165
CEGUI::WindowProperties::UnifiedPosition (Property to access the unified position of the window ) . . . . .	1167
CEGUI::WindowProperties::UnifiedSize (Property to access the unified position of the window )	1169

CEGUI::WindowProperties::UnifiedWidth (Property to access the unified width of the window )	1171
CEGUI::WindowProperties::UnifiedXPosition (Property to access the unified position x-coordinate of the window )	1173
CEGUI::WindowProperties::UnifiedYPosition (Property to access the unified position y-coordinate of the window )	1175
CEGUI::UnknownObjectException (Exception class used when a request was made using a name of an unknown object )	1177
CEGUI::UpdateEventArgs (WindowEventArgs class that is primarily used by lua scripts )	1179
CEGUI::URect (Area rectangle class built using unified dimensions (UDims) )	1180
CEGUI::UVector2 (Two dimensional vector class built using unified dimensions (UDims). The UVector2 class is used for representing both positions and sizes )	1181
CEGUI::ComboboxProperties::ValidationString (Property to access the string used for regular expression validation of the edit box text )	1182
CEGUI::EditboxProperties::ValidationString (Property to access the string used for regular expression validation of the edit box text )	1184
CEGUI::Vector2 (Class used as a two dimensional vector (aka a Point) )	1186
CEGUI::Vector3 (Class used as a three dimensional vector )	1187
CEGUI::ThumbProperties::VertFree (Property to access the state the setting to free the thumb vertically )	1188
CEGUI::WindowProperties::VerticalAlignment (Property to access the vertical alignment setting for the window )	1190
CEGUI::ScrollablePaneProperties::VertOverlapSize (Property to access the overlap size for the vertical Scrollbar )	1192
CEGUI::ThumbProperties::VertRange (Property to access the vertical movement range for the thumb )	1194
CEGUI::ScrollablePaneProperties::VertScrollPosition (Property to access the scroll position of the vertical Scrollbar )	1196
CEGUI::ScrollablePaneProperties::VertStepSize (Property to access the step size for the vertical Scrollbar )	1198
CEGUI::WindowProperties::Visible (Property to access window Visible setting )	1200
CEGUI::WindowProperties::WantsMultiClickEvents (Property to control whether the window will receive double/triple-click events )	1202
CEGUI::WidgetComponent (Class that encapsulates information regarding a sub-widget required for a widget )	1204
CEGUI::WidgetDim (Dimension type that represents some dimension of a Window/widget. Implements BaseDim interface )	1206
CEGUI::WidgetLookFeel (Class that encapsulates look & feel information for a particular widget type )	1208
CEGUI::WidgetLookManager (Manager class that gives top-level access to widget data based "look and feel" specifications loaded into the system )	1217
CEGUI::Window (An abstract base class providing common functionality and specifying the required interface for derived classes )	1222
CEGUI::WindowEventArgs (EventArgs based class that is used for objects passed to handlers triggered for events concerning some Window object )	1298
CEGUI::WindowFactory (Abstract class that defines the required interface for all WindowFactory objects )	1299
CEGUI::WindowFactoryManager (Class that manages WindowFactory objects )	1301
CEGUI::WindowFactoryManager::AliasTargetStack (Class used to track active alias targets for Window factory types )	1308
CEGUI::WindowFactoryManager::FalagardWindowMapping (Struct used to hold mapping information required to create a falagard based window )	1309
CEGUI::WindowManager (The WindowManager class describes an object that manages creation and lifetime of Window objects )	1310
CEGUI::WindowRenderer (Base-class for the assignable WindowRenderer object )	1318

<a href="#">CEGUI::WindowProperties::WindowRenderer</a> (Property to access/change the assigned window renderer object ) . . . . .	1323
<a href="#">CEGUI::WindowRendererFactory</a> (Base-class for <a href="#">WindowRendererFactory</a> ) . . . . .	1325
<a href="#">CEGUI::MultiLineEditboxProperties::WordWrap</a> (Property to access the word-wrap setting of the edit box ) . . . . .	1326
<a href="#">CEGUI::XMLAttributes</a> (Class representing a block of attributes associated with an XML element ) . . . . .	1328
<a href="#">CEGUI::XMLParser</a> (This is an abstract class that is used by <a href="#">CEGUI</a> to interface with XML parser libraries ) . . . . .	1333
<a href="#">CEGUI::XMLSerializer</a> (Class used to create XML Document ) . . . . .	1336
<a href="#">CEGUI::WindowProperties::ZOrderChangeEnabled</a> (Property to access window Z-Order changing enabled setting ) . . . . .	1339



# Chapter 4

## Crazy Eddies GUI System Page Index

### 4.1 Crazy Eddies GUI System Related Pages

Here is a list of all related documentation pages:

Todo List . . . . .	<a href="#">1341</a>
Bug List . . . . .	<a href="#">1342</a>



## Chapter 5

# Crazy Eddies GUI System Namespace Documentation

### 5.1 CEGUI Namespace Reference

Main namespace for Crazy Eddie's GUI Library.

#### Classes

- class [BoundSlot](#)  
*Class that tracks a [SubscriberSlot](#), its group, and the [Event](#) to which it was subscribed. This is effectively what gets returned from the calls to the [Event::subscribe](#) members, though [BoundSlot](#) is always wrapped in a reference counted pointer. When a [BoundSlot](#) is deleted, the connection is unsubscribed and the [SubscriberSlot](#) is deleted.*
- class [colour](#)  
*Class representing [colour](#) values within the system.*
- class [ColourRect](#)  
*Class that holds details of colours for the four corners of a rectangle.*
- class [Config\\_xmlHandler](#)  
*Handler class used to parse the Configuration XML file.*
- class [CoordConverter](#)  
*Utility class that helps in converting various types of co-ordinate between absolute screen positions and positions offset from the top-left corner of a given [Window](#) object.*
- class [RawDataContainer](#)  
*Class used as the databuffer for loading files throughout the library.*
- class [DefaultLogger](#)  
*Default implementation for the [Logger](#) class. If you want to redirect [CEGUI](#) logs to some place other than a text file, implement your own [Logger](#) implementation and create a object of the [Logger](#) type before creating the [CEGUI::System](#) singleton.*

- class **DefaultResourceProvider**
- class [DynamicModule](#)  
*Class that wraps and gives access to a dynamically linked module (.dll, .so, etc...).*
- class [Event](#)  
*Defines an 'event' which can be subscribed to by interested parties.*
- class [EventArgs](#)  
*Base class used as the argument to all subscribers [Event](#) object.*
- class [EventSet](#)  
*Class that collects together a set of [Event](#) objects.*
- class [Exception](#)  
*Root exception class used within the GUI system.*
- class [GenericException](#)  
*[Exception](#) class used when none of the other classes are applicable.*
- class [UnknownObjectException](#)  
*[Exception](#) class used when a request was made using a name of an unknown object.*
- class [InvalidRequestException](#)  
*[Exception](#) class used when some impossible request was made for the current system state.*
- class [FileIOException](#)  
*[Exception](#) class used when a file handling problem occurs.*
- class [RendererException](#)  
*[Exception](#) class used when an problem is detected within the [Renderer](#) or related objects.*
- class [AlreadyExistsException](#)  
*[Exception](#) class used when an attempt is made to use an object name that is already in use within the system.*
- class [MemoryException](#)  
*[Exception](#) class used when a memory handling error is detected.*
- class [NullObjectException](#)  
*[Exception](#) class used when some required object or parameter is null.*
- class [ObjectInUseException](#)  
*[Exception](#) class used when some attempt to delete, remove, or otherwise invalidate some object that is still in use occurs.*
- class [ScriptException](#)  
*[Exception](#) class used when a scripting error occurs.*
- class [FactoryModule](#)

*Class that encapsulates access to a dynamic loadable module containing implementations of Windows, Widgets, and their factories.*

- class [FontGlyph](#)  
*internal class representing a single font glyph.*
- class [Font](#)  
*Class that encapsulates text rendering functionality for a typeface.*
- class [Font\\_xmlHandler](#)  
*Handler class used to parse the [Font](#) XML files using SAX2.*
- class [FontManager](#)  
*Class providing a shared library of [Font](#) objects to the system.*
- class [FreeFunctionSlot](#)  
*Slot functor class that calls back via a free function pointer.*
- class [FreeTypeFont](#)  
*Implementation of the [Font](#) class interface using the FreeType library.*
- class [FunctorCopySlot](#)  
*Slot template class that creates a functor that calls back via a copy of a functor object.*
- class [FunctorPointerSlot](#)  
*Slot template class that creates a functor that calls back via a functor object pointer.*
- struct [FunctorReferenceBinder](#)  
*Class that enables the creation of a reference binding for a functor object to be used as a callback slot. Wrap your functor with this to enable the use of an object reference when subscribing to an event signal (as opposed to the functor object being copied, or using a pointer).*
- class [FunctorReferenceSlot](#)  
*Slot template class that creates a functor that calls back via a functor object reference.*
- class [GlobalEventSet](#)  
*The [GlobalEventSet](#) singleton allows you to subscribe to an event for all instances of a class. The [GlobalEventSet](#) effectively supports "late binding" to events; which means you can subscribe to some event that does not actually exist (yet).*
- class [GUILayout\\_xmlHandler](#)  
*Handler class used to parse the GUILayout XML files using SAX2.*
- class [Image](#)  
*Class that represents a single [Image](#) of an [Imageset](#).*
- class [ImageCodec](#)  
*Abstract ImageLoader class. An image loader encapsulate the loading of a texture.*
- class [Imageset](#)

*Offers functions to define, access, and draw, a set of image components on a single graphical surface or Texture.*

- class [Imageset\\_xmlHandler](#)

*Handler class used to parse the [Imageset](#) XML files using SAX2.*

- class [ImagesetManager](#)

*Class providing a shared library of [Imageset](#) objects to the system.*

- struct [Key](#)

*struct to give scope to scan code enumeration.*

- class [WindowEventArgs](#)

*[EventArgs](#) based class that is used for objects passed to handlers triggered for events concerning some [Window](#) object.*

- class [UpdateEventArgs](#)

*[WindowEventArgs](#) class that is primarily used by lua scripts.*

- class [MouseEventArgs](#)

*[EventArgs](#) based class that is used for objects passed to input event handlers concerning mouse input.*

- class [MouseCursorEventArgs](#)

*[EventArgs](#) based class that is used for objects passed to input event handlers concerning mouse cursor events.*

- class [KeyEventArgs](#)

*[EventArgs](#) based class that is used for objects passed to input event handlers concerning keyboard input.*

- class [ActivationEventArgs](#)

*[EventArgs](#) based class that is used for Activated and Deactivated window events.*

- class [DragDropEventArgs](#)

*[EventArgs](#) based class used for certain drag/drop notifications.*

- class [ConstBaseIterator](#)

*Base class constant iterator used to offer iteration over various collections within the system.*

- class [Logger](#)

*Abstract class that defines the interface of a logger object for the GUI system. The default implementation of this interface is the [DefaultLogger](#) class; if you want to perform special logging, derive your own class from [Logger](#) and initialize a object of that type before you create the [CEGUI::System](#) singleton.*

- class [MemberFunctionSlot](#)

*Slot template class that creates a functor that calls back via a class member function.*

- class [MouseCursor](#)

*Class that allows access to the GUI system mouse cursor.*

- class [PixmapFont](#)

*Implementation of the [Font](#) class interface using static [Imageset](#)'s.*

- class [PropertyReceiver](#)  
*Dummy base class to ensure correct casting of receivers.*
- class [Property](#)  
*An abstract class that defines the interface to access object properties by name.*
- class [PropertyHelper](#)  
*Helper class used to convert various data types to and from the format expected in Property strings.*
- class [PropertySet](#)  
*Class that contains a collection of [Property](#) objects.*
- class [Rect](#)  
*Class encapsulating operations on a Rectangle.*
- class [RefCounted](#)  
*Simple, generic, reference counted pointer class. This is primarily here for use by the Events system to track when to delete slot bindings.*
- class [RenderCache](#)  
*Class that acts as a cache for images that need to be rendered.*
- class [Renderer](#)  
*Abstract class defining the interface for [Renderer](#) objects.*
- class [ResourceProvider](#)  
*Abstract class that defines the required interface for all resource provider sub-classes.*
- class [Scheme](#)  
*A class that groups a set of GUI elements and initialises the system to access those elements.*
- class [Scheme\\_xmlHandler](#)  
*Handler class used to parse the [Scheme](#) XML files using SAX2.*
- class [SchemeManager](#)  
*A class that manages the creation of, access to, and destruction of GUI [Scheme](#) objects.*
- class [ScriptModule](#)  
*Abstract interface required for all scripting support modules to be used with the [CEGUI](#) system.*
- class [ScriptFunctor](#)  
*Functor class used for binding named script functions to events.*
- class [ScriptWindowHelper](#)
- class **Singleton**
- class [Size](#)  
*Class that holds the size (width & height) of something.*
- class [SlotFunctorBase](#)

*Defines abstract interface which will be used when constructing various functor objects that bind slots to signals (or in [CEGUI](#) terms, handlers to events).*

- class [String](#)  
*String class used within the GUI system.*
- class [SubscriberSlot](#)  
*SubscriberSlot class which is used when subscribing to events.*
- class [System](#)  
*The System class is the CEGUI class that provides access to all other elements in this system.*
- class [Texture](#)  
*Abstract base class specifying the required interface for Texture objects.*
- class [TextUtils](#)  
*Text utility support class. This class is all static members. You do not create instances of this class.*
- class [UDim](#)  
*Class representing a unified dimension; that is a dimension that has both a relative 'scale' portion and and absolute 'offset' portion.*
- class [UVector2](#)  
*Two dimensional vector class built using unified dimensions (UDims). The UVector2 class is used for representing both positions and sizes.*
- class [URect](#)  
*Area rectangle class built using unified dimensions (UDims).*
- class [Vector2](#)  
*Class used as a two dimensional vector (aka a Point).*
- class [Vector3](#)  
*Class used as a three dimensional vector.*
- class [Window](#)  
*An abstract base class providing common functionality and specifying the required interface for derived classes.*
- class [WindowFactory](#)  
*Abstract class that defines the required interface for all WindowFactory objects.*
- class [WindowFactoryManager](#)  
*Class that manages WindowFactory objects.*
- class [WindowManager](#)  
*The WindowManager class describes an object that manages creation and lifetime of Window objects.*
- class [WindowRenderer](#)  
*Base-class for the assignable WindowRenderer object.*



- class [WindowRendererFactory](#)  
*Base-class for [WindowRendererFactory](#).*
- class **WindowRendererManager**
- class [XMLAttributes](#)  
*Class representing a block of attributes associated with an XML element.*
- class **XMLHandler**
- class [XMLParser](#)  
*This is an abstract class that is used by [CEGUI](#) to interface with XML parser libraries.*
- class [XMLSerializer](#)  
*Class used to create XML Document.*
- class [ButtonBase](#)  
*Base class for all the 'button' type widgets (push button, radio button, check-box, etc).*
- class [Checkbox](#)  
*Base class providing logic for Check-box widgets.*
- class [ClippedContainer](#)  
*Helper container window that has configurable clipping. Used by the [ItemListbox](#) widget.*
- class [Combobox](#)  
*Base class for the [Combobox](#) widget.*
- class [ComboDropList](#)  
*Base class for the combo box drop down list. This is a specialisation of the [Listbox](#) class.*
- class [DragContainer](#)  
*Generic drag & drop enabled window class.*
- class [EditboxWindowRenderer](#)  
*Base class for the [EditboxWindowRenderer](#) class.*
- class [Editbox](#)  
*Base class for an [Editbox](#) widget.*
- class [FrameWindow](#)  
*Abstract base class for a movable, sizable, window with a title-bar and a frame.*
- class [GroupBox](#)  
*Base class for standard [GroupBox](#) widget.*
- class [GUISheet](#)  
*[Window](#) class intended to be used as a simple, generic [Window](#).*
- class [ItemEntryWindowRenderer](#)  
*Base class for [ItemEntry](#) window renderer objects.*

- class [ItemEntry](#)  
*Base class for item type widgets.*
- class [ItemListBaseWindowRenderer](#)  
*Base class for [ItemListBase](#) window renderer.*
- class [ItemListBase](#)  
*Base class for item list widgets.*
- class [ItemListbox](#)  
*[ItemListbox](#) window class.*
- class [ListboxWindowRenderer](#)  
*Base class for [Listbox](#) window renderer.*
- class [Listbox](#)  
*Base class for standard [Listbox](#) widget.*
- class [ListboxItem](#)  
*Base class for list box items.*
- class [ListboxTextItem](#)  
*Class used for textual items in a list box.*
- class [HeaderSequenceEventArgs](#)  
*[EventArgs](#) class used for segment move (sequence changed) events.*
- class [ListHeaderWindowRenderer](#)  
*Base class for the multi column list header window renderer.*
- class [ListHeader](#)  
*Base class for the multi column list header widget.*
- class [ListHeaderSegment](#)  
*Base class for list header segment window.*
- class [Menubar](#)  
*Base class for menu bars.*
- class [MenuBase](#)  
*Abstract base class for menus.*
- class [MenuItem](#)  
*Base class for menu items.*
- struct [MCLGridRef](#)  
*Simple grid index structure.*
- class [MultiColumnListWindowRenderer](#)  
*Base class for the multi column list window renderer.*

- class [MultiColumnList](#)  
*Base class for the multi column list widget.*
- class [MultiLineEditboxWindowRenderer](#)  
*Base class for multi-line edit box window renderer objects.*
- class [MultiLineEditbox](#)  
*Base class for the multi-line edit box widget.*
- class [PopupMenu](#)  
*Base class for popup menus.*
- class [ProgressBar](#)  
*Base class for progress bars.*
- class [PushButton](#)  
*Base class to provide logic for push button type widgets.*
- class [RadioButton](#)  
*Base class to provide the logic for Radio Button widgets.*
- class [ScrollablePaneWindowRenderer](#)  
*Base class for [ScrollablePane](#) window renderer objects.*
- class [ScrollablePane](#)  
*Base class for the [ScrollablePane](#) widget.*
- class [ScrollbarWindowRenderer](#)  
*Base class for [ItemEntry](#) window renderer objects.*
- class [Scrollbar](#)  
*Base scroll bar class.*
- class [ScrolledContainer](#)  
*Helper container window class which is used in the implementation of the [ScrollablePane](#) widget class.*
- class [ScrolledItemListBase](#)  
*[ScrolledItemListBase](#) window class.*
- class [SliderWindowRenderer](#)  
*Base class for [ItemEntry](#) window renderer objects.*
- class [Slider](#)  
*Base class for [Slider](#) widgets.*
- class [Spinner](#)  
*Base class for the [Spinner](#) widget.*
- class [TabButton](#)

Base class for *TabButtons*. A *TabButton* based class is used internally as the button that appears at the top of a *TabControl* widget to select the active tab pane.

- class *TabControlWindowRenderer*  
*Base class for TabControl window renderer objects.*
- class *TabControl*  
*Base class for standard Tab Control widget.*
- class *Thumb*  
*Base class for Thumb widget.*
- class *Titlebar*  
*Class representing the title bar for Frame Windows.*
- class *TooltipWindowRenderer*  
*Base class for Tooltip window renderer objects.*
- class *Tooltip*  
*Base class for Tooltip widgets.*
- class *TreeEventArgs*  
*EventArgs based class that is used for objects passed to input event handlers concerning Tree events.*
- class *Tree*  
*Base class for standard Tree widget.*
- class *TreeItem*  
*Base class for list box items.*
- class *Falagard\_xmlHandler*  
*Handler class used to parse look & feel XML files used by the Falagard system.*
- class *FalagardComponentBase*  
*Common base class used for renderable components within an ImagerySection.*
- class *BaseDim*  
*Abstract interface for a generic 'dimension' class.*
- class *AbsoluteDim*  
*Dimension type that represents an absolute pixel value. Implements BaseDim interface.*
- class *ImageDim*  
*Dimension type that represents some dimension of a named Image. Implements BaseDim interface.*
- class *WidgetDim*  
*Dimension type that represents some dimension of a Window/widget. Implements BaseDim interface.*
- class *UnifiedDim*  
*Dimension type that represents an Unified dimension. Implements BaseDim interface.*

- class [FontDim](#)  
*Dimension type that represents some metric of a [Font](#). Implements [BaseDim](#) interface.*
- class [PropertyDim](#)  
*Dimension type that represents the value of a [Window](#) property. Implements [BaseDim](#) interface.*
- class [Dimension](#)  
*Class representing some kind of dimension.*
- class [ComponentArea](#)  
*Class that represents a target area for a widget or imagery component.*
- class [FrameComponent](#)  
*Class that encapsulates information for a frame with background (9 images in total).*
- class [ImageryComponent](#)  
*Class that encapsulates information for a single image component.*
- class [ImagerySection](#)  
*Class that encapsulates a re-usable collection of imagery specifications.*
- class [LayerSpecification](#)  
*Class that encapsulates a single layer of imagery.*
- class [NamedArea](#)  
*[NamedArea](#) defines an area for a component which may later be obtained and referenced by a name unique to the [WidgetLook](#) holding the [NamedArea](#).*
- class [PropertyDefinition](#)  
*Class representing a generic get/set property.*
- class [PropertyDefinitionBase](#)  
*common base class used for types representing a new property to be available on all widgets that use the [WidgetLook](#) that the property definition is a part of.*
- class [PropertyInitialiser](#)  
*Class that holds information about a property and it's required initial value.*
- class [PropertyLinkDefinition](#)  
*Class representing a property that links to another property defined on an attached child widget.*
- class [SectionSpecification](#)  
*Class that represents a simple 'link' to an [ImagerySection](#).*
- class [StateImagery](#)  
*Class the encapsulates imagery for a given widget state.*
- class [TextComponent](#)  
*Class that encapsulates information for a text component.*

- class [WidgetComponent](#)  
*Class that encapsulates information regarding a sub-widget required for a widget.*
- class [WidgetLookAndFeel](#)  
*Class that encapsulates look & feel information for a particular widget type.*
- class [WidgetLookAndFeelManager](#)  
*Manager class that gives top-level access to widget data based "look and feel" specifications loaded into the system.*
- class [FalagardXMLHelper](#)  
*Utility helper class primarily intended for use by the falagard xml parser.*
- class [SubComp](#)  
*Implementation helper functor which is used to locate a [BoundSlot](#) in the multimap collection of BoundSlots.*
- class [SimpleTimer](#)  
*Simple timer class.*
- struct [MouseClickedTracker](#)  
*Implementation structure used in tracking up & down mouse button inputs in order to generate click, double-click, and triple-click events.*
- struct [MouseClickedTrackerImpl](#)
- struct [RegexValidator](#)  
*Internal struct to contain compiled regex string.*

## Namespaces

- namespace [CheckboxProperties](#)  
*Namespace containing all classes that make up the properties interface for the [Checkbox](#) class.*
- namespace [ComboboxProperties](#)  
*Namespace containing all classes that make up the properties interface for the [Combobox](#) class.*
- namespace [EditboxProperties](#)  
*Namespace containing all classes that make up the properties interface for the [Editbox](#) class.*
- namespace [FontProperties](#)  
*Namespace containing all classes that make up the properties interface for the [Font](#) base class.*
- namespace [FrameWindowProperties](#)  
*Namespace containing all classes that make up the properties interface for the [FrameWindow](#) class.*
- namespace [ItemEntryProperties](#)  
*Namespace containing all classes that make up the properties interface for the [ItemEntry](#) class.*
- namespace [ItemListBaseProperties](#)

*Namespace containing all classes that make up the properties interface for the [ItemListBase](#) class.*

- namespace [ItemListboxProperties](#)

*Namespace containing all classes that make up the properties interface for the [ItemListbox](#) class.*

- namespace [ListboxProperties](#)

*Namespace containing all classes that make up the properties interface for the [Listbox](#) class.*

- namespace [ListHeaderProperties](#)

*Namespace containing all classes that make up the properties interface for the [ListHeader](#) class.*

- namespace [ListHeaderSegmentProperties](#)

*Namespace containing all classes that make up the properties interface for the [ListHeaderSegment](#) class.*

- namespace [MultiColumnListProperties](#)

*Namespace containing all classes that make up the properties interface for the [MultiColumnList](#) class.*

- namespace [MultiLineEditboxProperties](#)

*Namespace containing all classes that make up the properties interface for the [MultiLineEditbox](#) class.*

- namespace [ProgressBarProperties](#)

*Namespace containing all classes that make up the properties interface for the [ProgressBar](#) class.*

- namespace [RadioButtonProperties](#)

*Namespace containing all classes that make up the properties interface for the [RadioButton](#) class.*

- namespace [ScrollablePaneProperties](#)

*Namespace containing all classes that make up the properties interface for the [ScrollablePane](#) class.*

- namespace [ScrollbarProperties](#)

*Namespace containing all classes that make up the properties interface for the [Scrollbar](#) class.*

- namespace [ScrolledContainerProperties](#)

*Namespace containing all classes that make up the properties interface for the [ScrolledContainer](#) class.*

- namespace [ScrolledItemListBaseProperties](#)

*Namespace containing all classes that make up the properties interface for the [ScrolledItemListBase](#) class.*

- namespace [SliderProperties](#)

*Namespace containing all classes that make up the properties interface for the [Slider](#) class.*

- namespace [TabControlProperties](#)

*Namespace containing all classes that make up the properties interface for the [Listbox](#) class.*

- namespace [ThumbProperties](#)

*Namespace containing all classes that make up the properties interface for the [Thumb](#) class.*

- namespace [TitlebarProperties](#)

*Namespace containing all classes that make up the properties interface for the [Titlebar](#) class.*

- namespace [TooltipProperties](#)

*Namespace containing all classes that make up the properties interface for the [Tooltip](#) class.*

- namespace [TreeProperties](#)

*Namespace containing all classes that make up the properties interface for the [Listbox](#) class.*

- namespace [WindowProperties](#)

*Namespace containing all classes that make up the properties interface for the [Window](#) base class.*

## Typedefs

- typedef unsigned long **ulong**
- typedef unsigned short **ushort**
- typedef unsigned int **uint**
- typedef unsigned char **uchar**
- typedef unsigned int **uint32**
- typedef unsigned short **uint16**
- typedef unsigned char **uint8**
- typedef std::ostream [OutStream](#)

*Output stream class.*

- typedef uint32 [argb\\_t](#)

*32 bit ARGB representation of a [colour](#).*

- typedef uint8 **utf8**
- typedef uint32 **utf32**
- typedef [Vector2](#) [Point](#)

*Point class.*

- typedef [GUISheet](#) [DefaultWindow](#)

*typedef for [DefaultWindow](#), which is the new name for [GUISheet](#).*

## Enumerations

- enum [TextFormatting](#) {  
[LeftAligned](#), [RightAligned](#), [Centred](#), [Justified](#),  
[WordWrapLeftAligned](#), [WordWrapRightAligned](#), [WordWrapCentred](#), [WordWrapJustified](#) }  
*Enumerated type that contains valid formatting types that can be specified when rendering text into a [Rect](#) area (the formatting [Rect](#)).*
- enum [MouseButton](#) {  
[LeftButton](#), [RightButton](#), [MiddleButton](#), [X1Button](#),  
[X2Button](#), [MouseButtonCount](#), [NoButton](#) }
- enum [SystemKey](#) {  
[LeftMouse](#) = 0x0001, [RightMouse](#) = 0x0002, [Shift](#) = 0x0004, [Control](#) = 0x0008,  
[MiddleMouse](#) = 0x0010, [X1Mouse](#) = 0x0020, [X2Mouse](#) = 0x0040, [Alt](#) = 0x0080 }



*System key flag values.*

- enum `LoggingLevel` {  
`Errors`, `Warnings`, `Standard`, `Informative`,  
`Insane` }  
*Enumeration of logging levels.*
- enum `MouseCursorImage` { `BlankMouseCursor` = 0, `DefaultMouseCursor` = -1 }  
*Enumeration of special values used for mouse cursor settings in Window objects.*
- enum `OrientationFlags` { `FlipHorizontal` = 1, `FlipVertical` = 2, `RotateRightAngle` = 4 }  
*Enumerated type that contains the valid flags that can be to use when rendering image.*
- enum `QuadSplitMode` { `TopLeftToBottomRight`, `BottomLeftToTopRight` }  
*Enumerated type that contains the valid diagonal-mode that specify how a quad is split into triangles when rendered with fx. a 3D API.*
- enum `VerticalAlignment` { `VA_TOP`, `VA_CENTRE`, `VA_BOTTOM` }  
*Enumerated type used when specifying vertical alignments.*
- enum `HorizontalAlignment` { `HA_LEFT`, `HA_CENTRE`, `HA_RIGHT` }  
*Enumerated type used when specifying horizontal alignments.*
- enum `DimensionType` {  
`DT_LEFT_EDGE`, `DT_X_POSITION`, `DT_TOP_EDGE`, `DT_Y_POSITION`,  
`DT_RIGHT_EDGE`, `DT_BOTTOM_EDGE`, `DT_WIDTH`, `DT_HEIGHT`,  
`DT_X_OFFSET`, `DT_Y_OFFSET`, `DT_INVALID` }  
*Enumeration of possible values to indicate what a given dimension represents.*
- enum `VerticalFormatting` {  
`VF_TOP_ALIGNED`, `VF_CENTRE_ALIGNED`, `VF_BOTTOM_ALIGNED`, `VF_STRETCHED`,  
`VF_TILED` }  
*Enumeration of possible values to indicate the vertical formatting to be used for an image component.*
- enum `HorizontalFormatting` {  
`HF_LEFT_ALIGNED`, `HF_CENTRE_ALIGNED`, `HF_RIGHT_ALIGNED`, `HF_STRETCHED`,  
`HF_TILED` }  
*Enumeration of possible values to indicate the horizontal formatting to be used for an image component.*
- enum `VerticalTextFormatting` { `VTF_TOP_ALIGNED`, `VTF_CENTRE_ALIGNED`, `VTF_BOTTOM_ALIGNED` }  
*Enumeration of possible values to indicate the vertical formatting to be used for a text component.*
- enum `HorizontalTextFormatting` {  
`HTF_LEFT_ALIGNED`, `HTF_RIGHT_ALIGNED`, `HTF_CENTRE_ALIGNED`, `HTF_JUSTIFIED`,  
`HTF_WORDWRAP_LEFT_ALIGNED`, `HTF_WORDWRAP_RIGHT_ALIGNED`, `HTF_WORDWRAP_CENTRE_ALIGNED`, `HTF_WORDWRAP_JUSTIFIED` }

*Enumeration of possible values to indicate the horizontal formatting to be used for a text component.*

- enum `FontMetricType` { `FMT_LINE_SPACING`, `FMT_BASELINE`, `FMT_HORZ_EXTENT` }

*Enumeration of possible values to indicate a particular font metric.*

- enum `DimensionOperator` {  
`DOP_NOOP`, `DOP_ADD`, `DOP_SUBTRACT`, `DOP_MULTIPLY`,  
`DOP_DIVIDE` }

*Enumeration of values representing mathematical operations on dimensions.*

- enum `FrameImageComponent` {  
`FIC_BACKGROUND`, `FIC_TOP_LEFT_CORNER`, `FIC_TOP_RIGHT_CORNER`, `FIC_BOTTOM_LEFT_CORNER`,  
`FIC_BOTTOM_RIGHT_CORNER`, `FIC_LEFT_EDGE`, `FIC_RIGHT_EDGE`, `FIC_TOP_EDGE`,  
`FIC_BOTTOM_EDGE`, `FIC_FRAME_IMAGE_COUNT` }

*Enumeration of values referencing available images forming a frame component.*

## Functions

- bool CEGUIEXPORT `operator==` (const `String` &str1, const `String` &str2)  
*Return true if `String` str1 is equal to `String` str2.*
- bool CEGUIEXPORT `operator==` (const `String` &str, const std::string &std\_str)  
*Return true if `String` str is equal to std::string std\_str.*
- bool CEGUIEXPORT `operator==` (const std::string &std\_str, const `String` &str)  
*Return true if `String` str is equal to std::string std\_str.*
- bool CEGUIEXPORT `operator==` (const `String` &str, const utf8 \*utf8\_str)  
*Return true if `String` str is equal to null-terminated utf8 data utf8\_str.*
- bool CEGUIEXPORT `operator==` (const utf8 \*utf8\_str, const `String` &str)  
*Return true if `String` str is equal to null-terminated utf8 data utf8\_str.*
- bool CEGUIEXPORT `operator!=` (const `String` &str1, const `String` &str2)  
*Return true if `String` str1 is not equal to `String` str2.*
- bool CEGUIEXPORT `operator!=` (const `String` &str, const std::string &std\_str)  
*Return true if `String` str is not equal to std::string std\_str.*
- bool CEGUIEXPORT `operator!=` (const std::string &std\_str, const `String` &str)  
*Return true if `String` str is not equal to std::string std\_str.*
- bool CEGUIEXPORT `operator!=` (const `String` &str, const utf8 \*utf8\_str)  
*Return true if `String` str is not equal to null-terminated utf8 data utf8\_str.*
- bool CEGUIEXPORT `operator!=` (const utf8 \*utf8\_str, const `String` &str)

Return true if *String* str is not equal to null-terminated utf8 data utf8\_str.

- bool CEGUIEXPORT operator< (const *String* &str1, const *String* &str2)  
Return true if *String* str1 is lexicographically less than *String* str2.
- bool CEGUIEXPORT operator< (const *String* &str, const std::string &std\_str)  
Return true if *String* str is lexicographically less than std::string std\_str.
- bool CEGUIEXPORT operator< (const std::string &std\_str, const *String* &str)  
Return true if *String* str is lexicographically less than std::string std\_str.
- bool CEGUIEXPORT operator< (const *String* &str, const utf8 \*utf8\_str)  
Return true if *String* str is lexicographically less than null-terminated utf8 data utf8\_str.
- bool CEGUIEXPORT operator< (const utf8 \*utf8\_str, const *String* &str)  
Return true if *String* str is lexicographically less than null-terminated utf8 data utf8\_str.
- bool CEGUIEXPORT operator> (const *String* &str1, const *String* &str2)  
Return true if *String* str1 is lexicographically greater than *String* str2.
- bool CEGUIEXPORT operator> (const *String* &str, const std::string &std\_str)  
Return true if *String* str is lexicographically greater than std::string std\_str.
- bool CEGUIEXPORT operator> (const std::string &std\_str, const *String* &str)  
Return true if *String* str is lexicographically greater than std::string std\_str.
- bool CEGUIEXPORT operator> (const *String* &str, const utf8 \*utf8\_str)  
Return true if *String* str is lexicographically greater than null-terminated utf8 data utf8\_str.
- bool CEGUIEXPORT operator> (const utf8 \*utf8\_str, const *String* &str)  
Return true if *String* str is lexicographically greater than null-terminated utf8 data utf8\_str.
- bool CEGUIEXPORT operator<= (const *String* &str1, const *String* &str2)  
Return true if *String* str1 is lexicographically less than or equal to *String* str2.
- bool CEGUIEXPORT operator<= (const *String* &str, const std::string &std\_str)  
Return true if *String* str is lexicographically less than or equal to std::string std\_str.
- bool CEGUIEXPORT operator<= (const std::string &std\_str, const *String* &str)  
Return true if *String* str is lexicographically less than or equal to std::string std\_str.
- bool CEGUIEXPORT operator<= (const *String* &str, const utf8 \*utf8\_str)  
Return true if *String* str is lexicographically less than or equal to null-terminated utf8 data utf8\_str.
- bool CEGUIEXPORT operator<= (const utf8 \*utf8\_str, const *String* &str)  
Return true if *String* str is lexicographically less than or equal to null-terminated utf8 data utf8\_str.
- bool CEGUIEXPORT operator>= (const *String* &str1, const *String* &str2)  
Return true if *String* str1 is lexicographically greater than or equal to *String* str2.

- `bool CEGUIEXPORT operator>= (const String &str, const std::string &std_str)`  
Return true if *String* str is lexicographically greater than or equal to *std::string* std\_str.
- `bool CEGUIEXPORT operator>= (const std::string &std_str, const String &str)`  
Return true if *String* str is lexicographically greater than or equal to *std::string* std\_str.
- `bool CEGUIEXPORT operator>= (const String &str, const utf8 *utf8_str)`  
Return true if *String* str is lexicographically greater than or equal to null-terminated utf8 data utf8\_str.
- `bool CEGUIEXPORT operator>= (const utf8 *utf8_str, const String &str)`  
Return true if *String* str is lexicographically greater than or equal to null-terminated utf8 data utf8\_str.
- `bool CEGUIEXPORT operator== (const String &str, const char *c_str)`  
Return true if *String* str is equal to c-string c\_str.
- `bool CEGUIEXPORT operator== (const char *c_str, const String &str)`  
Return true if c-string c\_str is equal to *String* str.
- `bool CEGUIEXPORT operator!= (const String &str, const char *c_str)`  
Return true if *String* str is not equal to c-string c\_str.
- `bool CEGUIEXPORT operator!= (const char *c_str, const String &str)`  
Return true if c-string c\_str is not equal to *String* str.
- `bool CEGUIEXPORT operator< (const String &str, const char *c_str)`  
Return true if *String* str is lexicographically less than c-string c\_str.
- `bool CEGUIEXPORT operator< (const char *c_str, const String &str)`  
Return true if c-string c\_str is lexicographically less than *String* str.
- `bool CEGUIEXPORT operator> (const String &str, const char *c_str)`  
Return true if *String* str is lexicographically greater than c-string c\_str.
- `bool CEGUIEXPORT operator> (const char *c_str, const String &str)`  
Return true if c-string c\_str is lexicographically greater than *String* str.
- `bool CEGUIEXPORT operator<= (const String &str, const char *c_str)`  
Return true if *String* str is lexicographically less than or equal to c-string c\_str.
- `bool CEGUIEXPORT operator<= (const char *c_str, const String &str)`  
Return true if c-string c\_str is lexicographically less than or equal to *String* str.
- `bool CEGUIEXPORT operator>= (const String &str, const char *c_str)`  
Return true if *String* str is lexicographically greater than or equal to c-string c\_str.
- `bool CEGUIEXPORT operator>= (const char *c_str, const String &str)`  
Return true if c-string c\_str is lexicographically greater than or equal to *String* str.
- `String CEGUIEXPORT operator+ (const String &str1, const String &str2)`  
Return *String* object that is the concatenation of the given inputs.

- **String** CEGUIEXPORT **operator+** (const **String** &str, const std::string &std\_str)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (const std::string &std\_str, const **String** &str)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (const **String** &str, const utf8 \*utf8\_str)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (const utf8 \*utf8\_str, const **String** &str)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (const **String** &str, utf32 code\_point)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (utf32 code\_point, const **String** &str)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (const **String** &str, const char \*c\_str)  
*Return **String** object that is the concatenation of the given inputs.*
- **String** CEGUIEXPORT **operator+** (const char \*c\_str, const **String** &str)  
*Return **String** object that is the concatenation of the given inputs.*
- CEGUIEXPORT std::ostream & **operator**<< (std::ostream &s, const **String** &str)
- void CEGUIEXPORT **swap** (**String** &str1, **String** &str2)  
*Swap the contents for two **String** objects.*
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**GUISheet**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**DragContainer**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ScrolledContainer**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ClippedContainer**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**Checkbox**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ComboDropList**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**Combobox**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**Editbox**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**FrameWindow**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ItemEntry**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ListHeader**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ListHeaderSegment**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**Listbox**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**MenuItem**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**Menubar**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**MultiColumnList**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**MultiLineEditbox**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**PopupMenu**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**ProgressBar**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**PushButton**)
- **CEGUI\_DECLARE\_WINDOW\_FACTORY** (**RadioButton**)

- CEGUI\_DECLARE\_WINDOW\_FACTORY ([ScrollablePane](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([Scrollbar](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([Slider](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([Spinner](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([TabButton](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([TabControl](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([Thumb](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([Titlebar](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([Tooltip](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([ItemListbox](#))
- CEGUI\_DECLARE\_WINDOW\_FACTORY ([GroupBox](#))
- bool [lbi\\_less](#) (const [ListboxItem](#) \*a, const [ListboxItem](#) \*b)

*Helper function used in sorting to compare two list box item text strings via the [ListboxItem](#) pointers and return if a is less than b.*

- bool [lbi\\_greater](#) (const [ListboxItem](#) \*a, const [ListboxItem](#) \*b)

*Helper function used in sorting to compare two list box item text strings via the [ListboxItem](#) pointers and return if a is greater than b.*

- bool [lbi\\_less](#) (const [TreeItem](#) \*a, const [TreeItem](#) \*b)

*Helper function used in sorting to compare two list box item text strings via the [TreeItem](#) pointers and return if a is less than b.*

- bool [lbi\\_greater](#) (const [TreeItem](#) \*a, const [TreeItem](#) \*b)

*Helper function used in sorting to compare two list box item text strings via the [TreeItem](#) pointers and return if a is greater than b.*

- CEGUI\_DEFINE\_WINDOW\_FACTORY ([GUISheet](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([DragContainer](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ScrolledContainer](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ClippedContainer](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Checkbox](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ComboDropList](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Combobox](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Editbox](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([FrameWindow](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ItemEntry](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ListHeader](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ListHeaderSegment](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Listbox](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([MenuItem](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Menubar](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([MultiColumnList](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([MultiLineEditbox](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([PopupMenu](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ProgressBar](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([PushButton](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([RadioButton](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([ScrollablePane](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Scrollbar](#))
- CEGUI\_DEFINE\_WINDOW\_FACTORY ([Slider](#))

- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([Spinner](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([TabButton](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([TabControl](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([Thumb](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([Titlebar](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([Tooltip](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([ItemListbox](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([GroupBox](#))
- **CEGUI\_DEFINE\_WINDOW\_FACTORY** ([Tree](#))
- static bool **ItemEntry\_less** (const [ItemEntry](#) \*a, const [ItemEntry](#) \*b)
- static bool **ItemEntry\_greater** (const [ItemEntry](#) \*a, const [ItemEntry](#) \*b)
- CFBundleRef **mac\_loadExeBundle** (const char \*name)
- void \* **mac\_getBundleSym** (CFBundleRef bundle, const char \*name)
- bool **mac\_unloadExeBundle** (CFBundleRef bundle)
- const char \* **mac\_errorBundle** ()

## Variables

- static const float [DefaultNativeHorzRes](#) = 640.0f  
*Default native horizontal resolution (for fonts and imagesets).*
- static const float [DefaultNativeVertRes](#) = 480.0f  
*Default native vertical resolution (for fonts and imagesets).*
- static const [String](#) **FontNameAttribute** ("Name")
- static const [String](#) **FontFilenameAttribute** ("Filename")
- static const [String](#) **FontResourceGroupAttribute** ("ResourceGroup")
- static const [String](#) **FontAutoScaledAttribute** ("AutoScaled")
- static const [String](#) **FontNativeHorzResAttribute** ("NativeHorzRes")
- static const [String](#) **FontNativeVertResAttribute** ("NativeVertRes")
- static const [String](#) **FontElement** ("Font")
- static const [String](#) **MappingElement** ("Mapping")
- static const [String](#) **FontTypeAttribute** ("Type")
- static const [String](#) **FontSchemaName** ("Font.xsd")
- static const [String](#) **FontTypeFreeType** ("FreeType")
- static const [String](#) **FontTypePixmap** ("Pixmap")
- static int **ft\_usage\_count** = 0
- static FT\_Library **ft\_lib**
- static const [String](#) **FontSizeAttribute** ("Size")
- static const [String](#) **FontAntiAliasedAttribute** ("AntiAlias")
- static const [String](#) **BuiltInResourceGroup** ("\*")
- static const [String](#) **MappingCodepointAttribute** ("Codepoint")
- static const [String](#) **MappingImageAttribute** ("Image")
- static const [String](#) **MappingHorzAdvanceAttribute** ("HorzAdvance")
- static const [String](#) **EnableTop** = "EnableTop"
- static const [String](#) **EnableBottom** = "EnableBottom"
- static const [String](#) **n0** = "0"
- static const [String](#) **n1** = "1"

### 5.1.1 Detailed Description

Main namespace for Crazy Eddie's GUI Library.

The [CEGUI](#) namespace contains all the classes and other items that comprise the core of Crazy Eddie's GUI system.

### 5.1.2 Enumeration Type Documentation

#### 5.1.2.1 enum CEGUI::DimensionOperator

Enumeration of values representing mathematical operations on dimensions.

**Enumerator:**

- DOP\_NOOP* Do nothing operator.
- DOP\_ADD* Dims should be added.
- DOP\_SUBTRACT* Dims should be subtracted.
- DOP\_MULTIPLY* Dims should be multiplied.
- DOP\_DIVIDE* Dims should be divided.

#### 5.1.2.2 enum CEGUI::DimensionType

Enumeration of possible values to indicate what a given dimension represents.

**Enumerator:**

- DT\_LEFT\_EDGE* [Dimension](#) represents the left edge of some entity (same as DT\_X\_POSITION).
- DT\_X\_POSITION* [Dimension](#) represents the x position of some entity (same as DT\_LEFT\_EDGE).
- DT\_TOP\_EDGE* [Dimension](#) represents the top edge of some entity (same as DT\_Y\_POSITION).
- DT\_Y\_POSITION* [Dimension](#) represents the y position of some entity (same as DT\_TOP\_EDGE).
- DT\_RIGHT\_EDGE* [Dimension](#) represents the right edge of some entity.
- DT\_BOTTOM\_EDGE* [Dimension](#) represents the bottom edge of some entity.
- DT\_WIDTH* [Dimension](#) represents the width of some entity.
- DT\_HEIGHT* [Dimension](#) represents the height of some entity.
- DT\_X\_OFFSET* [Dimension](#) represents the x offset of some entity (usually only applies to an [Image](#) entity).
- DT\_Y\_OFFSET* [Dimension](#) represents the y offset of some entity (usually only applies to an [Image](#) entity).
- DT\_INVALID* Invalid / uninitialised DimensionType.



### 5.1.2.3 enum CEGUI::FontMetricType

Enumeration of possible values to indicate a particular font metric.

**Enumerator:**

*FMT\_LINE\_SPACING* Vertical line spacing value for font.

*FMT\_BASELINE* Vertical baseline value for font.

*FMT\_HORZ\_EXTENT* Horizontal extent of a string.

### 5.1.2.4 enum CEGUI::FrameImageComponent

Enumeration of values referencing available images forming a frame component.

**Enumerator:**

*FIC\_BACKGROUND* References image used for the background.

*FIC\_TOP\_LEFT\_CORNER* References image used for the top-left corner.

*FIC\_TOP\_RIGHT\_CORNER* References image used for the top-right corner.

*FIC\_BOTTOM\_LEFT\_CORNER* References image used for the bottom-left corner.

*FIC\_BOTTOM\_RIGHT\_CORNER* References image used for the bottom-right corner.

*FIC\_LEFT\_EDGE* References image used for the left edge.

*FIC\_RIGHT\_EDGE* References image used for the right edge.

*FIC\_TOP\_EDGE* References image used for the top edge.

*FIC\_BOTTOM\_EDGE* References image used for the bottom edge.

*FIC\_FRAME\_IMAGE\_COUNT* Max number of images for a frame.

### 5.1.2.5 enum CEGUI::HorizontalAlignment

Enumerated type used when specifying horizontal alignments.

**Enumerator:**

*HA\_LEFT* Window's position specifies an offset of it's left edge from the left edge of it's parent.

*HA\_CENTRE* Window's position specifies an offset of it's horizontal centre from the horizontal centre of it's parent.

*HA\_RIGHT* Window's position specifies an offset of it's right edge from the right edge of it's parent.

### 5.1.2.6 enum CEGUI::HorizontalFormatting

Enumeration of possible values to indicate the horizontal formatting to be used for an image component.

**Enumerator:**

*HF\_LEFT\_ALIGNED* Left of [Image](#) should be aligned with the left of the destination area.

*HF\_CENTRE\_ALIGNED* [Image](#) should be horizontally centred within the destination area.

*HF\_RIGHT\_ALIGNED* Right of [Image](#) should be aligned with the right of the destination area.

*HF\_STRETCHED* [Image](#) should be stretched horizontally to fill the destination area.

*HF\_TILED* [Image](#) should be tiled horizontally to fill the destination area (right-most tile may be clipped).

### 5.1.2.7 enum CEGUI::HorizontalTextFormatting

Enumeration of possible values to indicate the horizontal formatting to be used for a text component.

**Enumerator:**

- HTF\_LEFT\_ALIGNED*** Left of text should be aligned with the left of the destination area (single line of text only).
- HTF\_RIGHT\_ALIGNED*** Right of text should be aligned with the right of the destination area (single line of text only).
- HTF\_CENTRE\_ALIGNED*** text should be horizontally centred within the destination area (single line of text only).
- HTF\_JUSTIFIED*** text should be spaced so that it takes the full width of the destination area (single line of text only).
- HTF\_WORDWRAP\_LEFT\_ALIGNED*** Left of text should be aligned with the left of the destination area (word wrapped to multiple lines as needed).
- HTF\_WORDWRAP\_RIGHT\_ALIGNED*** Right of text should be aligned with the right of the destination area (word wrapped to multiple lines as needed).
- HTF\_WORDWRAP\_CENTRE\_ALIGNED*** text should be horizontally centred within the destination area (word wrapped to multiple lines as needed).
- HTF\_WORDWRAP\_JUSTIFIED*** text should be spaced so that it takes the full width of the destination area (word wrapped to multiple lines as needed).

### 5.1.2.8 enum CEGUI::LoggingLevel

Enumeration of logging levels.

**Enumerator:**

- Errors*** Only actual error conditions will be logged.
- Warnings*** Warnings will be logged as well.
- Standard*** Basic events will be logged (default level).
- Informative*** Useful tracing (object creations etc) information will be logged.
- Insane*** Mostly everything gets logged (use for heavy tracing only, log WILL be big).

### 5.1.2.9 enum CEGUI::MouseButton

/brief Enumeration of mouse buttons

**Enumerator:**

- NoButton*** Value set for no mouse button. NB: This is not 0, do not assume!

### 5.1.2.10 enum CEGUI::MouseCursorImage

Enumeration of special values used for mouse cursor settings in [Window](#) objects.

**Enumerator:**

- BlankMouseCursor*** No image should be displayed for the mouse cursor.
- DefaultMouseCursor*** The default mouse cursor image should be displayed.

### 5.1.2.11 enum CEGUI::OrientationFlags

Enumerated type that contains the valid flags that can be to use when rendering image.

**Enumerator:**

*FlipHorizontal* Horizontal flip the image.

*FlipVertical* Vertical flip the image.

*RotateRightAngle* Rotate the image anticlockwise 90 degree.

### 5.1.2.12 enum CEGUI::QuadSplitMode

Enumerated type that contains the valid diagonal-mode that specify how a quad is split into triangles when rendered with fx. a 3D API.

**Enumerator:**

*TopLeftToBottomRight* Diagonal goes from top-left to bottom-right.

*BottomLeftToTopRight* Diagonal goes from bottom-left to top-right.

### 5.1.2.13 enum CEGUI::SystemKey

[System](#) key flag values.

**Enumerator:**

*LeftMouse* The left mouse button.

*RightMouse* The right mouse button.

*Shift* Either shift key.

*Control* Either control key.

*MiddleMouse* The middle mouse button.

*X1Mouse* The first 'extra' mouse button.

*X2Mouse* The second 'extra' mouse button.

*Alt* Either alt key.

### 5.1.2.14 enum CEGUI::TextFormatting

Enumerated type that contains valid formatting types that can be specified when rendering text into a [Rect](#) area (the formatting [Rect](#)).

**Enumerator:**

*LeftAligned* All text is printed on a single line. The left-most character is aligned with the left edge of the formatting [Rect](#).

*RightAligned* All text is printed on a single line. The right-most character is aligned with the right edge of the formatting [Rect](#).

*Centred* All text is printed on a single line. The text is centred horizontally in the formatting [Rect](#).

***Justified*** All text is printed on a single line. The left-most and right-most characters are aligned with the edges of the formatting [Rect](#).

***WordWrapLeftAligned*** Text is broken into multiple lines no wider than the formatting [Rect](#). The left-most character of each line is aligned with the left edge of the formatting [Rect](#).

***WordWrapRightAligned*** Text is broken into multiple lines no wider than the formatting [Rect](#). The right-most character of each line is aligned with the right edge of the formatting [Rect](#).

***WordWrapCentred*** Text is broken into multiple lines no wider than the formatting [Rect](#). Each line is centred horizontally in the formatting [Rect](#).

***WordWrapJustified*** Text is broken into multiple lines no wider than the formatting [Rect](#). The left-most and right-most characters of each line are aligned with the edges of the formatting [Rect](#).

#### 5.1.2.15 enum CEGUI::VerticalAlignment

Enumerated type used when specifying vertical alignments.

##### Enumerator:

***VA\_TOP*** Window's position specifies an offset of it's top edge from the top edge of it's parent.

***VA\_CENTRE*** Window's position specifies an offset of it's vertical centre from the vertical centre of it's parent.

***VA\_BOTTOM*** Window's position specifies an offset of it's bottom edge from the bottom edge of it's parent.

#### 5.1.2.16 enum CEGUI::VerticalFormatting

Enumeration of possible values to indicate the vertical formatting to be used for an image component.

##### Enumerator:

***VF\_TOP\_ALIGNED*** Top of [Image](#) should be aligned with the top of the destination area.

***VF\_CENTRE\_ALIGNED*** [Image](#) should be vertically centred within the destination area.

***VF\_BOTTOM\_ALIGNED*** Bottom of [Image](#) should be aligned with the bottom of the destination area.

***VF\_STRETCHED*** [Image](#) should be stretched vertically to fill the destination area.

***VF\_TILED*** [Image](#) should be tiled vertically to fill the destination area (bottom-most tile may be clipped).

#### 5.1.2.17 enum CEGUI::VerticalTextFormatting

Enumeration of possible values to indicate the vertical formatting to be used for a text component.

##### Enumerator:

***VTF\_TOP\_ALIGNED*** Top of text should be aligned with the top of the destination area.

***VTF\_CENTRE\_ALIGNED*** text should be vertically centred within the destination area.

***VTF\_BOTTOM\_ALIGNED*** Bottom of text should be aligned with the bottom of the destination area.

### 5.1.3 Function Documentation

#### 5.1.3.1 String CEGUI::operator+ (const char \* *c\_str*, const String & *str*)

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*c\_str* c-string describing the first part of the new string  
*str* [String](#) object describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *c\_str* and *str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

#### 5.1.3.2 String CEGUI::operator+ (const String & *str*, const char \* *c\_str*)

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*str* [String](#) object describing first part of the new string  
*c\_str* c-string describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *str* and *c\_str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

#### 5.1.3.3 String CEGUI::operator+ (utf32 *code\_point*, const String & *str*)

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*code\_point* utf32 code point describing the first part of the new string  
*str* [String](#) object describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *code\_point* and *str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.4 String CEGUI::operator+ (const String & *str*, utf32 *code\_point*)**

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*str* [String](#) object describing the first part of the new string

*code\_point* utf32 code point describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *str* and *code\_point*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.5 String CEGUI::operator+ (const utf8 \* *utf8\_str*, const String & *str*)**

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the first part of the new string

*str* [String](#) object describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *str* and *utf8\_str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.6 String CEGUI::operator+ (const String & *str*, const utf8 \* *utf8\_str*)**

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*str* [String](#) object describing first part of the new string

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *str* and *utf8\_str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.7 String CEGUI::operator+ (const std::string & *std\_str*, const String & *str*)**

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*std\_str* std::string object describing the first part of the new string

*str* [String](#) object describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *std\_str* and *str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.8 String CEGUI::operator+ (const String & *str*, const std::string & *std\_str*)**

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*str* [String](#) object describing first part of the new string

*std\_str* std::string object describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *str* and *std\_str*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.9 String CEGUI::operator+ (const String & *str1*, const String & *str2*)**

Return [String](#) object that is the concatenation of the given inputs.

**Parameters:**

*str1* [String](#) object describing first part of the new string

*str2* [String](#) object describing the second part of the new string

**Returns:**

A [String](#) object that is the concatenation of *str1* and *str2*

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**5.1.3.10 void CEGUI::swap (String & *str1*, String & *str2*)**

Swap the contents for two [String](#) objects.

**Parameters:**

*str1* [String](#) object who's contents are to be swapped with *str2*

*str2* [String](#) object who's contents are to be swapped with *str1*

**Returns:**

Nothing



## 5.2 CEGUI::CheckboxProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Checkbox](#) class.

### Classes

- class [Selected](#)  
*Property to access the selected state of the check box.*

### 5.2.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Checkbox](#) class.

## 5.3 CEGUI::ComboboxProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Combobox](#) class.

### Classes

- class [ReadOnly](#)  
*Property to access the read-only setting of the edit box.*
- class [ValidationString](#)  
*Property to access the string used for regular expression validation of the edit box text.*
- class [CaratIndex](#)  
*Property to access the current carat index.*
- class [EditSelectionStart](#)  
*Property to access the current selection start index.*
- class [EditSelectionLength](#)  
*Property to access the current selection length.*
- class [MaxEditTextLength](#)  
*Property to access the maximum text length for the edit box.*
- class [SortList](#)  
*Property to access the sort setting of the list box.*
- class [ForceVertScrollbar](#)  
*Property to access the 'always show' setting for the vertical scroll bar of the list box.*
- class [ForceHorzScrollbar](#)  
*Property to access the 'always show' setting for the horizontal scroll bar of the list box.*
- class [SingleClickMode](#)  
*Property to access the 'single click mode' setting for the combo box.*

### 5.3.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Combobox](#) class.

## 5.4 CEGUI::EditboxProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Editbox](#) class.

### Classes

- class [ReadOnly](#)  
*Property to access the read-only setting of the edit box.*
- class [MaskText](#)  
*Property to access the mask text setting of the edit box.*
- class [MaskCodepoint](#)  
*Property to access the mask text setting of the edit box.*
- class [ValidationString](#)  
*Property to access the string used for regular expression validation of the edit box text.*
- class [CaratIndex](#)  
*Property to access the current carat index.*
- class [SelectionStart](#)  
*Property to access the current selection start index.*
- class [SelectionLength](#)  
*Property to access the current selection length.*
- class [MaxTextLength](#)  
*Property to access the maximum text length for the edit box.*
- class [NormalTextColour](#)  
*Property to access the normal, unselected, text colour used for rendering text.*
- class [SelectedTextColour](#)  
*Property to access the colour used for rendering text within the selection area.*
- class [ActiveSelectionColour](#)  
*Property to access the colour used for rendering the selection highlight when the edit box is active.*
- class [InactiveSelectionColour](#)  
*Property to access the colour used for rendering the selection highlight when the edit box is inactive.*

### 5.4.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Editbox](#) class.

## 5.5 CEGUI::FontProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Font](#) base class.

### Classes

- class **NativeRes**
- class **Name**
- class **FileName**
- class **ResourceGroup**
- class **AutoScaled**
- class **FreeTypePointSize**
- class **FreeTypeAntialiased**
- class **PixmapImageset**
- class **PixmapMapping**

### 5.5.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Font](#) base class.

## 5.6 CEGUI::FrameWindowProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [FrameWindow](#) class.

### Classes

- class [SizingEnabled](#)  
*Property to access the state of the sizable setting for the [FrameWindow](#).*
- class [FrameEnabled](#)  
*Property to access the setting for whether the window frame will be displayed.*
- class [TitlebarEnabled](#)  
*Property to access the setting for whether the window title-bar will be enabled (or displayed depending upon choice of final widget type).*
- class [CloseButtonEnabled](#)  
*Property to access the setting for whether the window close button will be enabled (or displayed depending upon choice of final widget type).*
- class [RollUpEnabled](#)  
*Property to access the setting for whether the user is able to roll-up / shade the window.*
- class [RollUpState](#)  
*Property to access the roll-up / shade state of the window.*
- class [DragMovingEnabled](#)  
*Property to access the setting for whether the user may drag the window around by its title bar.*
- class [SizingBorderThickness](#)  
*Property to access the setting for the sizing border thickness.*
- class [NSSizingCursorImage](#)  
*Property to access the N-S (up-down) sizing cursor image.*
- class [EWSizingCursorImage](#)  
*Property to access the E-W (left/right) sizing cursor image.*
- class [NWSEsizingCursorImage](#)  
*Property to access the NW-SE diagonal sizing cursor image.*
- class [NESWSizingCursorImage](#)  
*Property to access the NE-SW diagonal sizing cursor image.*

### 5.6.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [FrameWindow](#) class.

## 5.7 CEGUI::ItemEntryProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ItemEntry](#) class.

### Classes

- class [Selectable](#)  
*Property to access the state of the selectable setting.*
- class [Selected](#)  
*Property to access the state of the selected setting.*

### 5.7.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ItemEntry](#) class.

## 5.8 CEGUI::ItemListBaseProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ItemListBase](#) class.

### Classes

- class [AutoResizeEnabled](#)  
*Property to access the state of the auto resize enabled setting.*
- class [SortEnabled](#)  
*Property to access the state of the sorting enabled setting.*
- class [SortMode](#)  
*Property to access the sorting mode.*

### 5.8.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ItemListBase](#) class.

## 5.9 CEGUI::ItemListboxProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ItemListbox](#) class.

### Classes

- class [MultiSelect](#)  
*Property to access the state of the multiselect enabled setting.*

### 5.9.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ItemListbox](#) class.



## 5.10 CEGUI::ListboxProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Listbox](#) class.

### Classes

- class [Sort](#)  
*Property to access the sort setting of the list box.*
- class [MultiSelect](#)  
*Property to access the multi-select setting of the list box.*
- class [ForceVertScrollbar](#)  
*Property to access the 'always show' setting for the vertical scroll bar of the list box.*
- class [ForceHorzScrollbar](#)  
*Property to access the 'always show' setting for the horizontal scroll bar of the list box.*
- class [ItemTooltips](#)  
*Property to access the show item tooltips setting of the list box.*

### 5.10.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Listbox](#) class.

## 5.11 CEGUI::ListHeaderProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ListHeader](#) class.

### Classes

- class [ColumnsSizable](#)  
*Property to access the setting for user sizing of the column headers.*
- class [ColumnsMovable](#)  
*Property to access the setting for user moving of the column headers.*
- class [SortSettingEnabled](#)  
*Property to access the setting for user modification of the sort column & direction.*
- class [SortDirection](#)  
*Property to access the sort direction setting of the list header.*
- class [SortColumnID](#)  
*Property to access the current sort column (via ID code).*

### 5.11.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ListHeader](#) class.

## 5.12 CEGUI::ListHeaderSegmentProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ListHeaderSegment](#) class.

### Classes

- class [Sizable](#)  
*Property to access the sizable setting of the header segment.*
- class [Clickable](#)  
*Property to access the click-able setting of the header segment.*
- class [Dragable](#)  
*Property to access the drag-able setting of the header segment.*
- class [SortDirection](#)  
*Property to access the sort direction setting of the header segment.*
- class [SizingCursorImage](#)  
*Property to access the segment sizing cursor image.*
- class [MovingCursorImage](#)  
*Property to access the segment moving cursor image.*

### 5.12.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ListHeaderSegment](#) class.

## 5.13 CEGUI::MultiColumnListProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [MultiColumnList](#) class.

### Classes

- class [ColumnsSizable](#)  
*Property to access the setting for user sizing of the column headers.*
- class [ColumnsMovable](#)  
*Property to access the setting for user moving of the column headers.*
- class [SortSettingEnabled](#)  
*Property to access the setting for user modification of the sort column & direction.*
- class [SortDirection](#)  
*Property to access the sort direction setting of the list.*
- class [SortColumnID](#)  
*Property to access the current sort column (via ID code).*
- class [NominatedSelectionColumnID](#)  
*Property to access the nominated selection column (via ID).*
- class [NominatedSelectionRow](#)  
*Property to access the nominated selection row.*
- class [ForceVertScrollbar](#)  
*Property to access the 'always show' setting for the vertical scroll bar of the list box.*
- class [ForceHorzScrollbar](#)  
*Property to access the 'always show' setting for the horizontal scroll bar of the list box.*
- class [SelectionMode](#)  
*Property to access the selection mode setting of the list.*
- class [ColumnHeader](#)  
*Property to access a column.*
- class [RowCount](#)  
*Property to access the number of rows in the list (read-only).*

### 5.13.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [MultiColumnList](#) class.

## 5.14 CEGUI::MultiLineEditboxProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [MultiLineEditbox](#) class.

### Classes

- class [ReadOnly](#)  
*Property to access the read-only setting of the edit box.*
- class [WordWrap](#)  
*Property to access the word-wrap setting of the edit box.*
- class [CaratIndex](#)  
*Property to access the current carat index.*
- class [SelectionStart](#)  
*Property to access the current selection start index.*
- class [SelectionLength](#)  
*Property to access the current selection length.*
- class [MaxTextLength](#)  
*Property to access the maximum text length for the edit box.*
- class [SelectionBrushImage](#)  
*Property to access the selection brush image.*
- class [ForceVertScrollbar](#)  
*Property to access the 'always show' setting for the vertical scroll bar of the box.*

### 5.14.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [MultiLineEditbox](#) class.

## 5.15 CEGUI::ProgressBarProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ProgressBar](#) class.

### Classes

- class [CurrentProgress](#)  
*Property to access the current progress of the progress bar.*
- class [StepSize](#)  
*Property to access the step size setting for the progress bar.*

### 5.15.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ProgressBar](#) class.

## 5.16 CEGUI::RadioButtonProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [RadioButton](#) class.

### Classes

- class [Selected](#)  
*Property to access the selected state of the [RadioButton](#).*
- class [GroupID](#)  
*Property to access the radio button group ID.*

### 5.16.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [RadioButton](#) class.

## 5.17 CEGUI::ScrollablePaneProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ScrollablePane](#) class.

### Classes

- class [ContentPaneAutoSized](#)  
*Property to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content.*
- class [ContentArea](#)  
*Property to access the current content pane area rectangle (as window relative pixels).*
- class [ForceVertScrollbar](#)  
*Property to access the setting which controls whether the vertical scroll bar will always be displayed, or only displayed when it is required.*
- class [ForceHorzScrollbar](#)  
*Property to access the setting which controls whether the horizontal scroll bar will always be displayed, or only displayed when it is required.*
- class [HorzStepSize](#)  
*Property to access the step size for the horizontal [Scrollbar](#).*
- class [HorzOverlapSize](#)  
*Property to access the overlap size for the horizontal [Scrollbar](#).*
- class [HorzScrollPosition](#)  
*Property to access the scroll position of the horizontal [Scrollbar](#).*
- class [VertStepSize](#)  
*Property to access the step size for the vertical [Scrollbar](#).*
- class [VertOverlapSize](#)  
*Property to access the overlap size for the vertical [Scrollbar](#).*
- class [VertScrollPosition](#)  
*Property to access the scroll position of the vertical [Scrollbar](#).*

### 5.17.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ScrollablePane](#) class.



## 5.18 CEGUI::ScrollbarProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Scrollbar](#) class.

### Classes

- class [DocumentSize](#)  
*Property to access the document size for the [Scrollbar](#).*
- class [PageSize](#)  
*Property to access the page size for the [Scrollbar](#).*
- class [StepSize](#)  
*Property to access the step size for the [Scrollbar](#).*
- class [OverlapSize](#)  
*Property to access the overlap size for the [Scrollbar](#).*
- class [ScrollPosition](#)  
*Property to access the scroll position of the [Scrollbar](#).*

### 5.18.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Scrollbar](#) class.

## 5.19 CEGUI::ScrolledContainerProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ScrolledContainer](#) class.

### Classes

- class [ContentPaneAutoSized](#)  
*Property to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content.*
- class [ContentArea](#)  
*Property to access the current content pane area rectangle (as window relative pixels).*
- class [ChildExtentsArea](#)  
*Property offering read-only access to the current content extents rectangle (as window relative pixels).*

### 5.19.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ScrolledContainer](#) class.

## 5.20 CEGUI::ScrolledItemListBaseProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [ScrolledItemListBase](#) class.

### Classes

- class [ForceVertScrollbar](#)  
*Property to access the state of the force vertical scrollbar setting.*
- class [ForceHorzScrollbar](#)  
*Property to access the state of the force horizontal scrollbar setting.*

### 5.20.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [ScrolledItemListBase](#) class.

## 5.21 CEGUI::SliderProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Slider](#) class.

### Classes

- class [CurrentValue](#)  
*Property to access the current value of the slider.*
- class [MaximumValue](#)  
*Property to access the maximum value of the slider.*
- class [ClickStepSize](#)  
*Property to access the click-step size for the slider.*

### 5.21.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Slider](#) class.

## 5.22 CEGUI::TabControlProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Listbox](#) class.

### Classes

- class [TabHeight](#)  
*Property to access the tab height setting of the tab control.*
- class [TabTextPadding](#)  
*Property to access the tab text padding setting of the tab control.*
- class [TabPanePosition](#)  
*Property to query/set the position of the button pane in tab control.*

### 5.22.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Listbox](#) class.

## 5.23 CEGUI::ThumbProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Thumb](#) class.

### Classes

- class [HotTracked](#)  
*Property to access the state of the "hot-tracked" setting for the thumb.*
- class [VertFree](#)  
*Property to access the state the setting to free the thumb vertically.*
- class [HorzFree](#)  
*Property to access the state the setting to free the thumb horizontally.*
- class [VertRange](#)  
*Property to access the vertical movement range for the thumb.*
- class [HorzRange](#)  
*Property to access the horizontal movement range for the thumb.*

### 5.23.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Thumb](#) class.

## 5.24 CEGUI::TitlebarProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Titlebar](#) class.

### Classes

- class [DraggingEnabled](#)  
*Property to access the state of the dragging enabled setting for the [Titlebar](#).*

### 5.24.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Titlebar](#) class.

## 5.25 CEGUI::TooltipProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Tooltip](#) class.

### Classes

- class [HoverTime](#)  
*Property to access the timeout that must expire before the tooltip gets activated.*
- class [DisplayTime](#)  
*Property to access the time after which the tooltip automatically de-activates itself.*
- class [FadeTime](#)  
*Property to access the duration of the fade effect for the tooltip.*

### 5.25.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Tooltip](#) class.



## 5.26 CEGUI::TreeProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Listbox](#) class.

### Classes

- class [Sort](#)  
*Property to access the sort setting of the list box.*
- class [MultiSelect](#)  
*Property to access the multi-select setting of the list box.*
- class [ForceVertScrollbar](#)  
*Property to access the 'always show' setting for the vertical scroll bar of the list box.*
- class [ForceHorzScrollbar](#)  
*Property to access the 'always show' setting for the horizontal scroll bar of the list box.*
- class [ItemTooltips](#)  
*Property to access the show item tooltips setting of the list box.*

### 5.26.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Listbox](#) class.

## 5.27 CEGUI::WindowProperties Namespace Reference

Namespace containing all classes that make up the properties interface for the [Window](#) base class.

### Classes

- class [ID](#)  
*Property to access window [ID](#) field.*
- class [Alpha](#)  
*Property to access window alpha setting.*
- class [Font](#)  
*Property to access window [Font](#) setting.*
- class [Text](#)  
*Property to access window text setting.*
- class [MouseCursorImage](#)  
*Property to access window mouse cursor setting.*
- class [ClippedByParent](#)  
*Property to access window "clipped by parent" setting.*
- class [InheritsAlpha](#)  
*Property to access window "Inherits Alpha" setting.*
- class [AlwaysOnTop](#)  
*Property to access window "Always-On-Top" setting.*
- class [Disabled](#)  
*Property to access window [Disabled](#) setting.*
- class [Visible](#)  
*Property to access window [Visible](#) setting.*
- class [RestoreOldCapture](#)  
*Property to access window Restore Old Capture setting.*
- class [DestroyedByParent](#)  
*Property to access window Destroyed by Parent setting.*
- class [ZOrderChangeEnabled](#)  
*Property to access window Z-Order changing enabled setting.*
- class [WantsMultiClickEvents](#)  
*Property to control whether the window will receive double/triple-click events.*
- class [MouseButtonDownAutoRepeat](#)

*Property to control whether the window will receive autorepeat mouse button down events.*

- class [AutoRepeatDelay](#)

*Property to access window autorepeat delay value.*

- class [AutoRepeatRate](#)

*Property to access window autorepeat rate value.*

- class [DistributeCapturedInputs](#)

*Property to access whether inputs are passed to child windows when input is captured to this window.*

- class [CustomTooltipType](#)

*Property to access the custom tooltip for this [Window](#).*

- class [Tooltip](#)

*Property to access the tooltip text for this [Window](#).*

- class [InheritsTooltipText](#)

*Property to access whether the window inherits its tooltip text from its parent when it has no tooltip text of its own.*

- class [RiseOnClick](#)

*Property to access whether the window rises to the top of the z order when clicked.*

- class [VerticalAlignment](#)

*Property to access the vertical alignment setting for the window.*

- class [HorizontalAlignment](#)

*Property to access the horizontal alignment setting for the window.*

- class [UnifiedAreaRect](#)

*Property to access the unified area rectangle of the window.*

- class [UnifiedPosition](#)

*Property to access the unified position of the window.*

- class [UnifiedXPosition](#)

*Property to access the unified position x-coordinate of the window.*

- class [UnifiedYPosition](#)

*Property to access the unified position y-coordinate of the window.*

- class [UnifiedSize](#)

*Property to access the unified position of the window.*

- class [UnifiedWidth](#)

*Property to access the unified width of the window.*

- class [UnifiedHeight](#)

*Property to access the unified height of the window.*

- class [UnifiedMinSize](#)  
*Property to access the unified minimum size of the window.*
- class [UnifiedMaxSize](#)  
*Property to access the unified maximum size of the window.*
- class [MousePassThroughEnabled](#)  
*Property to access whether the window ignores mouse events and pass them through to any windows behind it.*
- class [WindowRenderer](#)  
*Property to access/change the assigned window renderer object.*
- class [LookNFeel](#)  
*Property to access/change the assigned look'n'feel.*
- class [DragDropTarget](#)  
*Property to get/set whether the [Window](#) will receive drag and drop related notifications.*

### 5.27.1 Detailed Description

Namespace containing all classes that make up the properties interface for the [Window](#) base class.

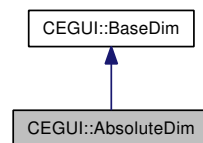
## Chapter 6

# Crazy Eddies GUI System Class Documentation

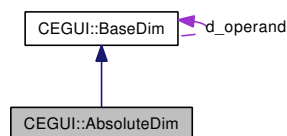
### 6.1 CEGUI::AbsoluteDim Class Reference

[Dimension](#) type that represents an absolute pixel value. Implements [BaseDim](#) interface.

Inheritance diagram for CEGUI::AbsoluteDim:



Collaboration diagram for CEGUI::AbsoluteDim:



#### Public Member Functions

- [AbsoluteDim](#) (float val)  
*Constructor.*
- void [setValue](#) (float val)  
*Set the current value of the [AbsoluteDim](#).*

#### Protected Member Functions

- float [getValue\\_impl](#) (const [Window](#) &wnd) const

*Implementataion method to return the base value for this [BaseDim](#). This method should not attempt to apply the mathematical operator; this is handled automatically.*

- float [getValue\\_impl](#) (const [Window](#) &wnd, const [Rect](#) &container) const

*Implementataion method to return the base value for this [BaseDim](#). This method should not attempt to apply the mathematical operator; this is handled automatically by [BaseDim](#).*

- void [writeXMLElementName\\_impl](#) ([XMLSerializer](#) &xml\_stream) const

*Implementataion method to output real xml element name.*

- void [writeXMLElementAttributes\\_impl](#) ([XMLSerializer](#) &xml\_stream) const

*Implementataion method to create the element attributes.*

- [BaseDim](#) \* [clone\\_impl](#) () const

*Implementataion method to return a clone of this sub-class of [BaseDim](#). This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by [BaseDim](#).*

### 6.1.1 Detailed Description

[Dimension](#) type that represents an absolute pixel value. Implements [BaseDim](#) interface.

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 CEGUI::AbsoluteDim::AbsoluteDim (float *val*)

Constructor.

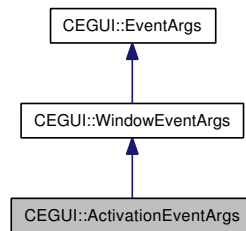
##### Parameters:

*val* float value to be assigned to the [AbsoluteDim](#).

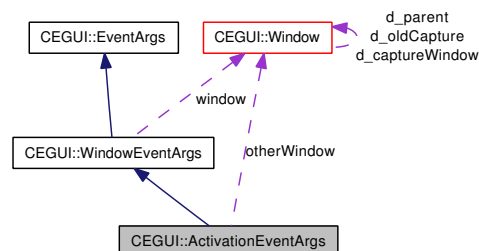
## 6.2 CEGUI::ActivationEventArgs Class Reference

[EventArgs](#) based class that is used for Activated and Deactivated window events.

Inheritance diagram for CEGUI::ActivationEventArgs:



Collaboration diagram for CEGUI::ActivationEventArgs:



### Public Member Functions

- [ActivationEventArgs](#) ([Window](#) \*wnd)

### Public Attributes

- [Window](#) \* [otherWindow](#)  
*Pointer to the other window involved in the activation change.*

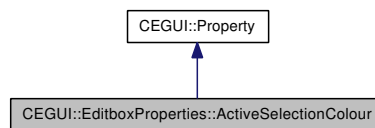
#### 6.2.1 Detailed Description

[EventArgs](#) based class that is used for Activated and Deactivated window events.

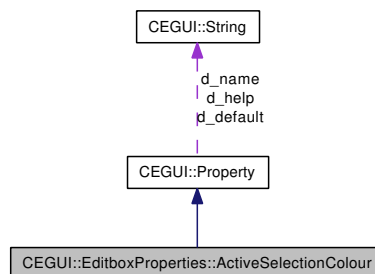
## 6.3 CEGUI::EditboxProperties::ActiveSelectionColour Class Reference

[Property](#) to access the [colour](#) used for rendering the selection highlight when the edit box is active.

Inheritance diagram for CEGUI::EditboxProperties::ActiveSelectionColour:



Collaboration diagram for CEGUI::EditboxProperties::ActiveSelectionColour:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.3.1 Detailed Description

[Property](#) to access the [colour](#) used for rendering the selection highlight when the edit box is active.

##### Usage:

- Name: [ActiveSelectionColour](#)
- Format: "aarrggbb".

##### Where:

- aarrggbb is the ARGB [colour](#) value to be used.



## 6.3.2 Member Function Documentation

### 6.3.2.1 String CEGUI::EditboxProperties::ActiveSelectionColour::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.3.2.2 void CEGUI::EditboxProperties::ActiveSelectionColour::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

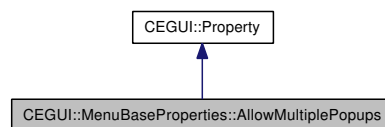
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

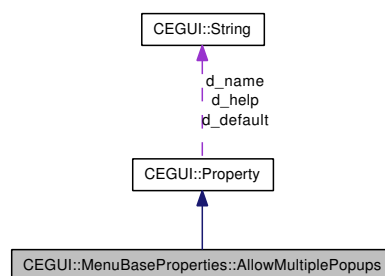
## 6.4 CEGUI::MenuBaseProperties::AllowMultiplePopups Class Reference

[Property](#) to access the state of the allow multiple popups setting.

Inheritance diagram for CEGUI::MenuBaseProperties::AllowMultiplePopups:



Collaboration diagram for CEGUI::MenuBaseProperties::AllowMultiplePopups:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.4.1 Detailed Description

[Property](#) to access the state of the allow multiple popups setting.

**Usage:**

- Name: [AllowMultiplePopups](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate that auto resizing is enabled.
- "False" to indicate that auto resizing is disabled.

## 6.4.2 Member Function Documentation

### 6.4.2.1 String CEGUI::MenuBaseProperties::AllowMultiplePopups::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.4.2.2 void CEGUI::MenuBaseProperties::AllowMultiplePopups::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

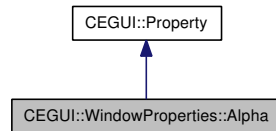
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

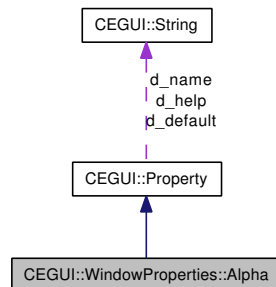
## 6.5 CEGUI::WindowProperties::Alpha Class Reference

[Property](#) to access window alpha setting.

Inheritance diagram for CEGUI::WindowProperties::Alpha:



Collaboration diagram for CEGUI::WindowProperties::Alpha:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.5.1 Detailed Description

[Property](#) to access window alpha setting.

This property offers access to the alpha-blend setting for the window.

#### Usage:

- Name: [Alpha](#)
- Format: "[float]".

#### Where:

- [float] is a floating point number between 0.0 and 1.0.

## 6.5.2 Member Function Documentation

### 6.5.2.1 String CEGUI::WindowProperties::Alpha::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.5.2.2 void CEGUI::WindowProperties::Alpha::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

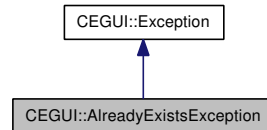
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

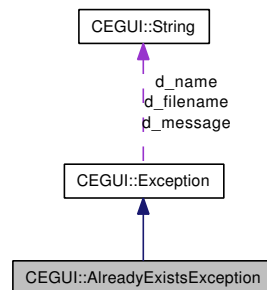
## 6.6 CEGUI::AlreadyExistsException Class Reference

**Exception** class used when an attempt is made to use an object name that is already in use within the system.

Inheritance diagram for CEGUI::AlreadyExistsException:



Collaboration diagram for CEGUI::AlreadyExistsException:



### Public Member Functions

- **AlreadyExistsException** (const **String** &message, const **String** &file="unknown", int line=0)  
*Constructor that is responsible for logging the already exists exception by calling the base class.*

### 6.6.1 Detailed Description

**Exception** class used when an attempt is made to use an object name that is already in use within the system.

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 CEGUI::AlreadyExistsException::AlreadyExistsException (const **String** &message, const **String** &file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the already exists exception by calling the base class.

#### Parameters:

- message** **String** object describing the reason for the already exists exception being thrown.
- filename** **String** object containing the name of the file where the already exists exception occurred.
- line** Integer representing the line number where the already exists exception occurred.

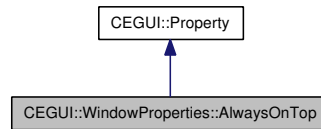
**Remarks:**

The already exists exception name is automatically passed to the base class as "CEGUI::AlreadyExistsException".

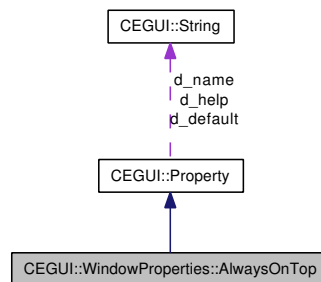
## 6.7 CEGUI::WindowProperties::AlwaysOnTop Class Reference

[Property](#) to access window "Always-On-Top" setting.

Inheritance diagram for CEGUI::WindowProperties::AlwaysOnTop:



Collaboration diagram for CEGUI::WindowProperties::AlwaysOnTop:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.7.1 Detailed Description

[Property](#) to access window "Always-On-Top" setting.

This property offers access to the always on top / topmost setting for the window.

##### Usage:

- Name: [AlwaysOnTop](#)
- Format: "[text]".

##### Where [Text] is:

- "True" to indicate the [Window](#) is always on top, and appears above all other non-always on top Windows.
- "False" to indicate the [Window](#) is not always on top, and will appear below all other always on top Windows.



## 6.7.2 Member Function Documentation

### 6.7.2.1 String CEGUI::WindowProperties::AlwaysOnTop::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.7.2.2 void CEGUI::WindowProperties::AlwaysOnTop::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

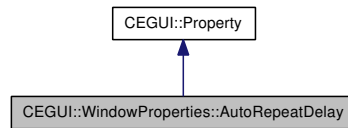
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

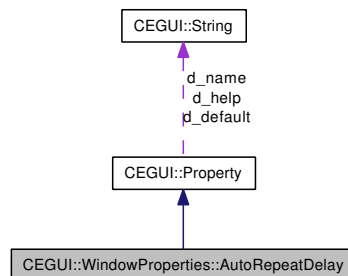
## 6.8 CEGUI::WindowProperties::AutoRepeatDelay Class Reference

[Property](#) to access window autorepeat delay value.

Inheritance diagram for CEGUI::WindowProperties::AutoRepeatDelay:



Collaboration diagram for CEGUI::WindowProperties::AutoRepeatDelay:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.8.1 Detailed Description

[Property](#) to access window autorepeat delay value.

This property offers access to the value that controls the initial delay for autorepeat mouse button down events.

#### Usage:

- Name: [AutoRepeatDelay](#)
- Format: "[float]".

#### Where:

- [float] specifies the delay in seconds.

## 6.8.2 Member Function Documentation

### 6.8.2.1 String CEGUI::WindowProperties::AutoRepeatDelay::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.8.2.2 void CEGUI::WindowProperties::AutoRepeatDelay::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

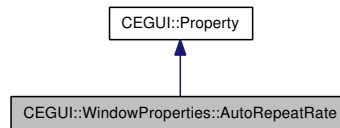
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

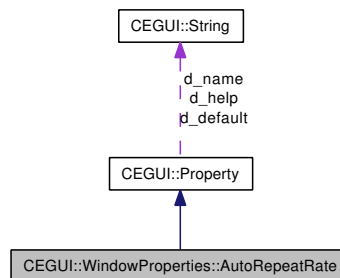
## 6.9 CEGUI::WindowProperties::AutoRepeatRate Class Reference

[Property](#) to access window autorepeat rate value.

Inheritance diagram for CEGUI::WindowProperties::AutoRepeatRate:



Collaboration diagram for CEGUI::WindowProperties::AutoRepeatRate:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.9.1 Detailed Description

[Property](#) to access window autorepeat rate value.

This property offers access to the value that controls the generation rate for autorepeat mouse button down events.

#### Usage:

- Name: [AutoRepeatRate](#)
- Format: "[float]".

#### Where:

- [float] specifies the rate at which autorepeat events will be generated in seconds.

## 6.9.2 Member Function Documentation

### 6.9.2.1 String CEGUI::WindowProperties::AutoRepeatRate::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.9.2.2 void CEGUI::WindowProperties::AutoRepeatRate::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

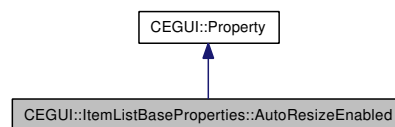
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

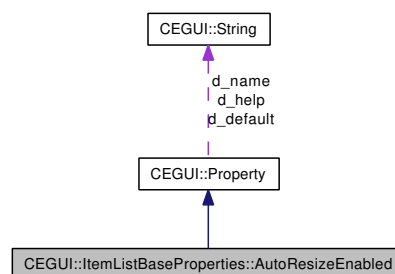
## 6.10 CEGUI::ItemListBaseProperties::AutoResizeEnabled Class Reference

[Property](#) to access the state of the auto resize enabled setting.

Inheritance diagram for CEGUI::ItemListBaseProperties::AutoResizeEnabled:



Collaboration diagram for CEGUI::ItemListBaseProperties::AutoResizeEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.10.1 Detailed Description

[Property](#) to access the state of the auto resize enabled setting.

**Usage:**

- Name: [AutoResizeEnabled](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate that auto resizing is enabled.
- "False" to indicate that auto resizing is disabled.

## 6.10.2 Member Function Documentation

### 6.10.2.1 String CEGUI::ItemListBaseProperties::AutoSizeEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.10.2.2 void CEGUI::ItemListBaseProperties::AutoSizeEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.11 CEGUI::BaseDim Class Reference

Abstract interface for a generic 'dimension' class.

Inherited by [CEGUI::AbsoluteDim](#), [CEGUI::FontDim](#), [CEGUI::ImageDim](#), [CEGUI::PropertyDim](#), [CEGUI::UnifiedDim](#), and [CEGUI::WidgetDim](#).

Collaboration diagram for CEGUI::BaseDim:



### Public Member Functions

- float [getValue](#) (const [Window](#) &wnd) const  
*Return a value that represents this dimension as absolute pixels.*
- float [getValue](#) (const [Window](#) &wnd, const [Rect](#) &container) const  
*Return a value that represents this dimension as absolute pixels.*
- [BaseDim](#) \* [clone](#) () const  
*Create an exact copy of the specialised class and return it as a pointer to a [BaseDim](#) object.*
- [DimensionOperator](#) [getDimensionOperator](#) () const  
*Return the [DimensionOperator](#) set for this [BaseDim](#) based object.*
- void [setDimensionOperator](#) ([DimensionOperator](#) op)  
*Set the [DimensionOperator](#) set for this [BaseDim](#) based object.*
- const [BaseDim](#) \* [getOperand](#) () const  
*Return a pointer to the [BaseDim](#) set to be used as the other operand.*
- void [setOperand](#) (const [BaseDim](#) &operand)  
*Set the [BaseDim](#) set to be used as the other operand in calculations for this [BaseDim](#).*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [BaseDim](#) to out\_stream.*

### Protected Member Functions

- virtual float [getValue\\_impl](#) (const [Window](#) &wnd) const =0  
*Implementataion method to return the base value for this [BaseDim](#). This method should not attempt to apply the mathematical operator; this is handled automatically.*
- virtual float [getValue\\_impl](#) (const [Window](#) &wnd, const [Rect](#) &container) const =0  
*Implementataion method to return the base value for this [BaseDim](#). This method should not attempt to apply the mathematical operator; this is handled automatically by [BaseDim](#).*
- virtual [BaseDim](#) \* [clone\\_impl](#) () const =0



*Implementataion method to return a clone of this sub-class of [BaseDim](#). This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by [BaseDim](#).*

- virtual void [writeXMLElementName\\_impl](#) (XMLSerializer &xml\_stream) const =0  
*Implementataion method to output real xml element name.*
- virtual void [writeXMLElementAttributes\\_impl](#) (XMLSerializer &xml\_stream) const =0  
*Implementataion method to create the element attributes.*

### 6.11.1 Detailed Description

Abstract interface for a generic 'dimension' class.

### 6.11.2 Member Function Documentation

#### 6.11.2.1 float CEGUI::BaseDim::getValue (const Window & wnd) const

Return a value that represents this dimension as absolute pixels.

##### Parameters:

**wnd** [Window](#) object that may be used by the specialised class to aid in calculating the final value.

##### Returns:

float value which represents, in pixels, the same value as this [BaseDim](#).

#### 6.11.2.2 float CEGUI::BaseDim::getValue (const Window & wnd, const Rect & container) const

Return a value that represents this dimension as absolute pixels.

##### Parameters:

**wnd** [Window](#) object that may be used by the specialised class to aid in calculating the final value (typically would be used to obtain window/widget dimensions).

**container** [Rect](#) object which describes an area to be considered as the base area when calculating the final value. Basically this means that relative values are calculated from the dimensions of this [Rect](#).

##### Returns:

float value which represents, in pixels, the same value as this [BaseDim](#).

#### 6.11.2.3 BaseDim \* CEGUI::BaseDim::clone () const

Create an exact copy of the specialised class and return it as a pointer to a [BaseDim](#) object.

Since the system needs to be able to copy objects derived from [BaseDim](#), but only has knowledge of the [BaseDim](#) interface, this clone method is provided to prevent slicing issues.

**Returns:**

[BaseDim](#) object pointer

**6.11.2.4 DimensionOperator CEGUI::BaseDim::getDimensionOperator () const**

Return the DimensionOperator set for this [BaseDim](#) based object.

**Returns:**

One of the DimensionOperator enumerated values representing a mathematical operation to be performed upon this [BaseDim](#) using the set operand.

**6.11.2.5 void CEGUI::BaseDim::setDimensionOperator (DimensionOperator *op*)**

Set the DimensionOperator set for this [BaseDim](#) based object.

**Parameters:**

*op* One of the DimensionOperator enumerated values representing a mathematical operation to be performed upon this [BaseDim](#) using the set operand.

**Returns:**

Nothing.

**6.11.2.6 const BaseDim \* CEGUI::BaseDim::getOperand () const**

Return a pointer to the [BaseDim](#) set to be used as the other operand.

**Returns:**

Pointer to the [BaseDim](#) object.

**6.11.2.7 void CEGUI::BaseDim::setOperand (const BaseDim & *operand*)**

Set the [BaseDim](#) set to be used as the other operand in calculations for this [BaseDim](#).

**Parameters:**

*operand* sub-class of [BaseDim](#) representing the 'other' operand. The given object will be cloned; no transfer of ownership occurs for the passed object.

**Returns:**

Nothing.

**6.11.2.8 void CEGUI::BaseDim::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [BaseDim](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

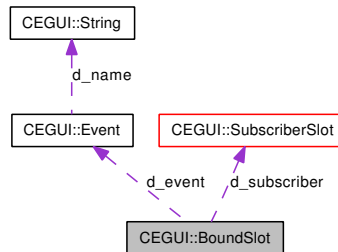
**Returns:**

Nothing.

## 6.12 CEGUI::BoundSlot Class Reference

Class that tracks a [SubscriberSlot](#), its group, and the [Event](#) to which it was subscribed. This is effectively what gets returned from the calls to the [Event::subscribe](#) members, though [BoundSlot](#) is always wrapped in a reference counted pointer. When a [BoundSlot](#) is deleted, the connection is unsubscribed and the [SubscriberSlot](#) is deleted.

Collaboration diagram for CEGUI::BoundSlot:



### Public Types

- typedef unsigned int **Group**

### Public Member Functions

- [BoundSlot](#) (Group group, const [SubscriberSlot](#) &subscriber, [Event](#) &event)  
*Constructor.*
- [BoundSlot](#) (const [BoundSlot](#) &other)  
*Copy constructor.*
- [~BoundSlot](#) ()  
*Destructor.*
- bool [connected](#) () const  
*Returns whether the slot which this object is tracking is still internally connected to the signal / event mechanism.*
- void [disconnect](#) ()  
*Disconnects the slot. Once disconnected, the slot will no longer be called when the associated signal / event fires. There is no way to re-connect a slot once it has been disconnected, a new subscription to the signal / event is required.*
- bool [operator==](#) (const [BoundSlot](#) &other) const  
*Equality operator.*
- bool [operator!=](#) (const [BoundSlot](#) &other) const  
*Non-equality operator.*

## Friends

- class **Event**

### 6.12.1 Detailed Description

Class that tracks a [SubscriberSlot](#), its group, and the [Event](#) to which it was subscribed. This is effectively what gets returned from the calls to the [Event::subscribe](#) members, though [BoundSlot](#) is always wrapped in a reference counted pointer. When a [BoundSlot](#) is deleted, the connection is unsubscribed and the [SubscriberSlot](#) is deleted.

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 CEGUI::BoundSlot::BoundSlot (Group *group*, const SubscriberSlot & *subscriber*, Event & *event*)

Constructor.

##### Parameters:

- group* The subscriber group this slot is attached to.
- subscriber* The actual slot object that is controlling this connection binding.
- event* The [Event](#) object to which the subscribed slot is attached.

### 6.12.3 Member Function Documentation

#### 6.12.3.1 bool CEGUI::BoundSlot::connected () const

Returns whether the slot which this object is tracking is still internally connected to the signal / event mechanism.

##### Returns:

- true to indicate that the slot is still connected.
- false to indicate that the slot has been disconnected.

#### 6.12.3.2 void CEGUI::BoundSlot::disconnect ()

Disconnects the slot. Once disconnected, the slot will no longer be called when the associated signal / event fires. There is no way to re-connect a slot once it has been disconnected, a new subscription to the signal / event is required.

##### Returns:

Nothing.

**6.12.3.3 bool CEGUI::BoundSlot::operator==(const BoundSlot & *other*) const**

Equality operator.

**Parameters:**

*other* The [BoundSlot](#) to compare against.

**Returns:**

- true if the [BoundSlot](#) objects represent the same connection.
- false if the [BoundSlot](#) objects represent different connections.

**6.12.3.4 bool CEGUI::BoundSlot::operator!=(const BoundSlot & *other*) const**

Non-equality operator.

**Parameters:**

*other* The [BoundSlot](#) to compare against.

**Returns:**

- true if the [BoundSlot](#) objects represent different connections.
- false if the [BoundSlot](#) objects represent the same connection.

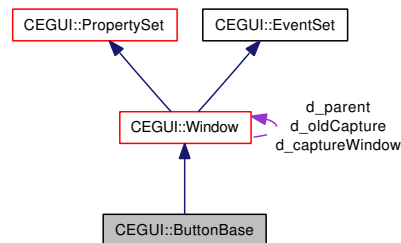
## 6.13 CEGUI::ButtonBase Class Reference

Base class for all the 'button' type widgets (push button, radio button, check-box, etc).

Inherits [CEGUI::Window](#).

Inherited by [CEGUI::Checkbox](#), [CEGUI::PushButton](#), [CEGUI::RadioButton](#), and [CEGUI::TabButton](#).

Collaboration diagram for CEGUI::ButtonBase:



### Public Member Functions

- bool [isHovering](#) (void) const  
*return true if user is hovering over this widget (or it's pushed and user is not over it for highlight)*
- bool [isPushed](#) (void) const  
*Return true if the button widget is in the pushed state.*
- [ButtonBase](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [ButtonBase](#) objects.*
- virtual [~ButtonBase](#) (void)  
*Destructor for [ButtonBase](#) objects.*

### Protected Member Functions

- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onMouseButtonDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseButtonUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void [onMouseLeaves](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has left this window's area.*

- void [updateInternalState](#) (const [Point](#) &mouse\_pos)  
*Update the internal state of the widget with the mouse at the given position.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

## Protected Attributes

- bool [d\\_pushed](#)  
*true when widget is pushed*
- bool [d\\_hovering](#)  
*true when the button is in 'hover' state and requires the hover rendering.*

### 6.13.1 Detailed Description

Base class for all the 'button' type widgets (push button, radio button, check-box, etc).

### 6.13.2 Member Function Documentation

#### 6.13.2.1 bool CEGUI::ButtonBase::isHovering (void) const [inline]

return true if user is hovering over this widget (or it's pushed and user is not over it for highlight)

##### Returns:

true if the user is hovering or if the button is pushed and the mouse is not over the button. Otherwise return false.

#### 6.13.2.2 bool CEGUI::ButtonBase::isPushed (void) const [inline]

Return true if the button widget is in the pushed state.

##### Returns:

true if the button-type widget is pushed, false if the widget is not pushed.

#### 6.13.2.3 void CEGUI::ButtonBase::onMouseMove (MouseEventArgs & e) [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

##### Parameters:

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::TabButton](#), and [CEGUI::Thumb](#).



#### 6.13.2.4 void CEGUI::ButtonBase::onMouseButtonDown (MouseEventArgs & *e*) [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

##### Parameters:

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::TabButton](#), and [CEGUI::Thumb](#).

#### 6.13.2.5 void CEGUI::ButtonBase::onMouseButtonUp (MouseEventArgs & *e*) [protected, virtual]

Handler called when a mouse button has been released within this window's area.

##### Parameters:

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::Checkbox](#), [CEGUI::PushButton](#), [CEGUI::RadioButton](#), and [CEGUI::TabButton](#).

#### 6.13.2.6 void CEGUI::ButtonBase::onCaptureLost (WindowEventArgs & *e*) [protected, virtual]

Handler called when this window loses capture of mouse inputs.

##### Parameters:

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::Thumb](#).

#### 6.13.2.7 void CEGUI::ButtonBase::onMouseLeaves (MouseEventArgs & *e*) [protected, virtual]

Handler called when the mouse cursor has left this window's area.

##### Parameters:

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.13.2.8 void CEGUI::ButtonBase::updateInternalState (const Point & *mouse\_pos*)**  
[protected]

Update the internal state of the widget with the mouse at the given position.

**Parameters:**

*mouse\_pos* Point object describing, in screen pixel co-ordinates, the location of the mouse cursor.

**Returns:**

Nothing

**6.13.2.9 virtual bool CEGUI::ButtonBase::testClassName\_impl (const String & *class\_name*) const**  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

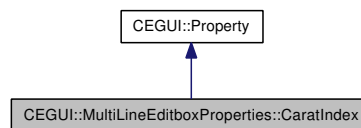
Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::Checkbox](#), [CEGUI::PushButton](#), [CEGUI::RadioButton](#), [CEGUI::TabButton](#), and [CEGUI::Thumb](#).

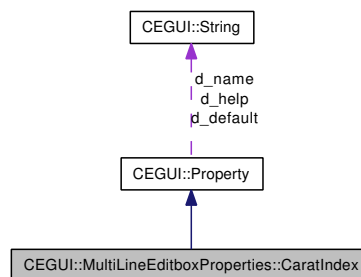
## 6.14 CEGUI::MultiLineEditboxProperties::CaratIndex Class Reference

[Property](#) to access the current carat index.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::CaratIndex:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::CaratIndex:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.14.1 Detailed Description

[Property](#) to access the current carat index.

##### Usage:

- Name: [CaratIndex](#)
- Format: "[uint]"

##### Where:

- [uint] is the zero based index of the carat position within the text.

## 6.14.2 Member Function Documentation

### 6.14.2.1 String CEGUI::MultiLineEditboxProperties::CaratIndex::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.14.2.2 void CEGUI::MultiLineEditboxProperties::CaratIndex::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

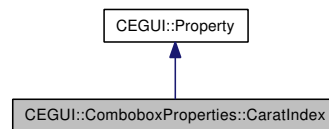
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

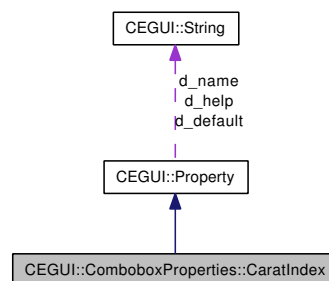
## 6.15 CEGUI::ComboboxProperties::CaratIndex Class Reference

[Property](#) to access the current carat index.

Inheritance diagram for CEGUI::ComboboxProperties::CaratIndex:



Collaboration diagram for CEGUI::ComboboxProperties::CaratIndex:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.15.1 Detailed Description

[Property](#) to access the current carat index.

#### Usage:

- Name: [CaratIndex](#)
- Format: "[uint]"

#### Where:

- [uint] is the zero based index of the carat position within the text.

## 6.15.2 Member Function Documentation

### 6.15.2.1 `String CEGUI::ComboboxProperties::CaratIndex::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.15.2.2 `void CEGUI::ComboboxProperties::CaratIndex::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

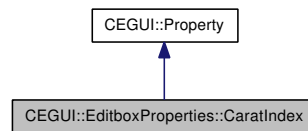
[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

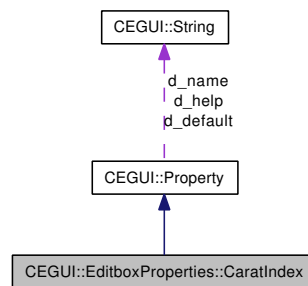
## 6.16 CEGUI::EditboxProperties::CaratIndex Class Reference

[Property](#) to access the current carat index.

Inheritance diagram for CEGUI::EditboxProperties::CaratIndex:



Collaboration diagram for CEGUI::EditboxProperties::CaratIndex:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.16.1 Detailed Description

[Property](#) to access the current carat index.

##### Usage:

- Name: [CaratIndex](#)
- Format: "[uint]"

##### Where:

- [uint] is the zero based index of the carat position within the text.

## 6.16.2 Member Function Documentation

### 6.16.2.1 `String CEGUI::EditboxProperties::CaratIndex::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.16.2.2 `void CEGUI::EditboxProperties::CaratIndex::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

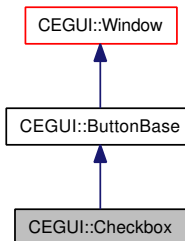
Implements [CEGUI::Property](#).



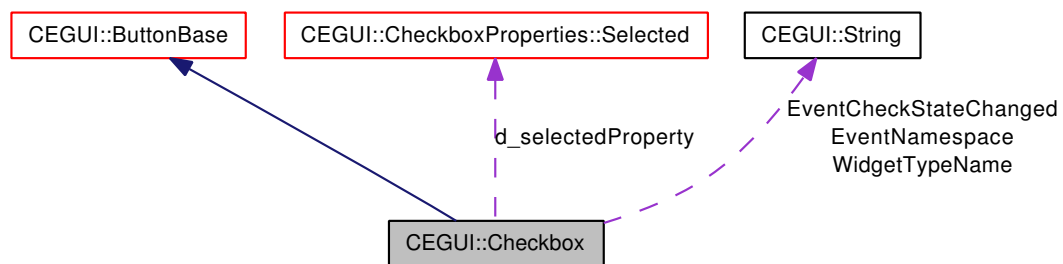
## 6.17 CEGUI::Checkbox Class Reference

Base class providing logic for Check-box widgets.

Inheritance diagram for CEGUI::Checkbox:



Collaboration diagram for CEGUI::Checkbox:



### Public Member Functions

- bool **isSelected** (void) const  
*return true if the check-box is selected (has the checkmark)*
- void **setSelected** (bool select)  
*set whether the check-box is selected or not*
- **Checkbox** (const **String** &type, const **String** &name)  
*Constructor for **Checkbox** class.*
- virtual **~Checkbox** (void)  
*Destructor for **Checkbox** class.*

### Static Public Attributes

- static const **String** **EventNamespace**  
*Namespace for global events.*
- static const **String** **WidgetTypeName**  
*Window factory name.*

- static const [String EventCheckStateChanged](#)

*The check-state of the widget has changed.*

## Protected Member Functions

- virtual void [onSelectStateChange](#) ([WindowEventArgs](#) &e)  
*event triggered internally when state of check-box changes*
- virtual void [onMouseButtonUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

## Protected Attributes

- bool [d\\_selected](#)  
*true if check-box is selected (has checkmark)*

### 6.17.1 Detailed Description

Base class providing logic for Check-box widgets.

### 6.17.2 Member Function Documentation

#### 6.17.2.1 **bool CEGUI::Checkbox::isSelected (void) const** [inline]

return true if the check-box is selected (has the checkmark)

##### Returns:

true if the widget is selected and has the check-mark, false if the widget is not selected and does not have the check-mark.

#### 6.17.2.2 **void CEGUI::Checkbox::setSelected (bool select)**

set whether the check-box is selected or not

##### Parameters:

*select* true to select the widget and give it the check-mark. false to de-select the widget and remove the check-mark.

##### Returns:

Nothing.

**6.17.2.3 void CEGUI::Checkbox::onMouseButtonUp (MouseEventArgs & *e*)** [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

**6.17.2.4 virtual bool CEGUI::Checkbox::testClassName\_impl (const String & *class\_name*) const** [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

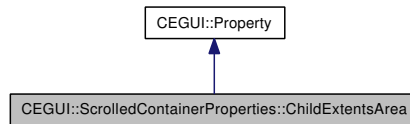
true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::ButtonBase](#).

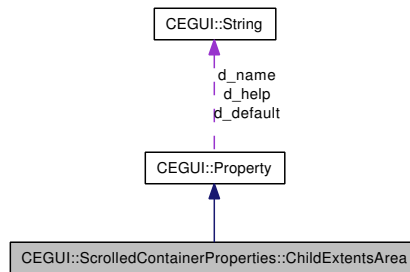
## 6.18 CEGUI::ScrolledContainerProperties::ChildExtentsArea Class Reference

[Property](#) offering read-only access to the current content extents rectangle (as window relative pixels).

Inheritance diagram for CEGUI::ScrolledContainerProperties::ChildExtentsArea:



Collaboration diagram for CEGUI::ScrolledContainerProperties::ChildExtentsArea:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.18.1 Detailed Description

[Property](#) offering read-only access to the current content extents rectangle (as window relative pixels).

##### Usage:

- Name: [ChildExtentsArea](#)
- Format: "l:[float] t:[float] r:[float] b:[float]".

##### Where:

- l:[float] specifies the position of the left edge of the area as a floating point number.
- t:[float] specifies the position of the top edge of the area as a floating point number.
- r:[float] specifies the position of the right edge of the area as a floating point number.
- b:[float] specifies the position of the bottom edge of the area as a floating point number.

## 6.18.2 Member Function Documentation

### 6.18.2.1 String CEGUI::ScrolledContainerProperties::ChildExtentsArea::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.18.2.2 void CEGUI::ScrolledContainerProperties::ChildExtentsArea::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

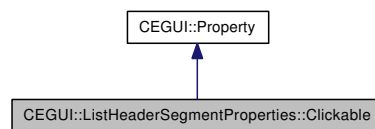
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

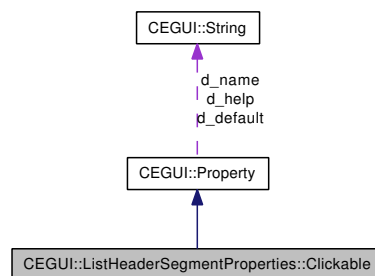
## 6.19 CEGUI::ListHeaderSegmentProperties::Clickable Class Reference

[Property](#) to access the click-able setting of the header segment.

Inheritance diagram for CEGUI::ListHeaderSegmentProperties::Clickable:



Collaboration diagram for CEGUI::ListHeaderSegmentProperties::Clickable:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.19.1 Detailed Description

[Property](#) to access the click-able setting of the header segment.

**Usage:**

- Name: [Clickable](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate the segment can be clicked by the user.
- "False" to indicate the segment can not be clicked by the user.

## 6.19.2 Member Function Documentation

### 6.19.2.1 String CEGUI::ListHeaderSegmentProperties::Clickable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.19.2.2 void CEGUI::ListHeaderSegmentProperties::Clickable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

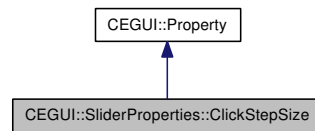
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

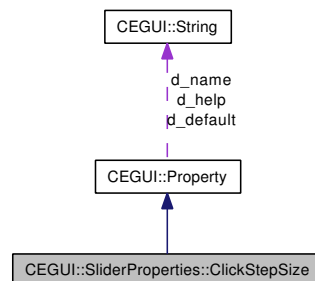
## 6.20 CEGUI::SliderProperties::ClickStepSize Class Reference

[Property](#) to access the click-step size for the slider.

Inheritance diagram for CEGUI::SliderProperties::ClickStepSize:



Collaboration diagram for CEGUI::SliderProperties::ClickStepSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.20.1 Detailed Description

[Property](#) to access the click-step size for the slider.

#### Usage:

- Name: [ClickStepSize](#)
- Format: "[float]".

#### Where:

- [float] represents the click-step size slider (this is how much the value changes when the slider container is clicked).



## 6.20.2 Member Function Documentation

### 6.20.2.1 String CEGUI::SliderProperties::ClickStepSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.20.2.2 void CEGUI::SliderProperties::ClickStepSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

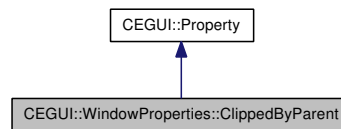
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

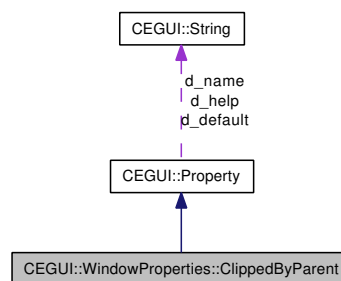
## 6.21 CEGUI::WindowProperties::ClippedByParent Class Reference

[Property](#) to access window "clipped by parent" setting.

Inheritance diagram for CEGUI::WindowProperties::ClippedByParent:



Collaboration diagram for CEGUI::WindowProperties::ClippedByParent:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.21.1 Detailed Description

[Property](#) to access window "clipped by parent" setting.

This property offers access to the clipped by parent setting for the window.

Usage:

- Name: [ClippedByParent](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the [Window](#) is clipped by it's parent.
- "False" to indicate the [Window](#) is not clipped by it's parent.

## 6.21.2 Member Function Documentation

### 6.21.2.1 String CEGUI::WindowProperties::ClippedByParent::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.21.2.2 void CEGUI::WindowProperties::ClippedByParent::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

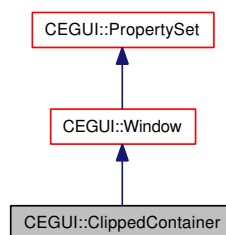
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

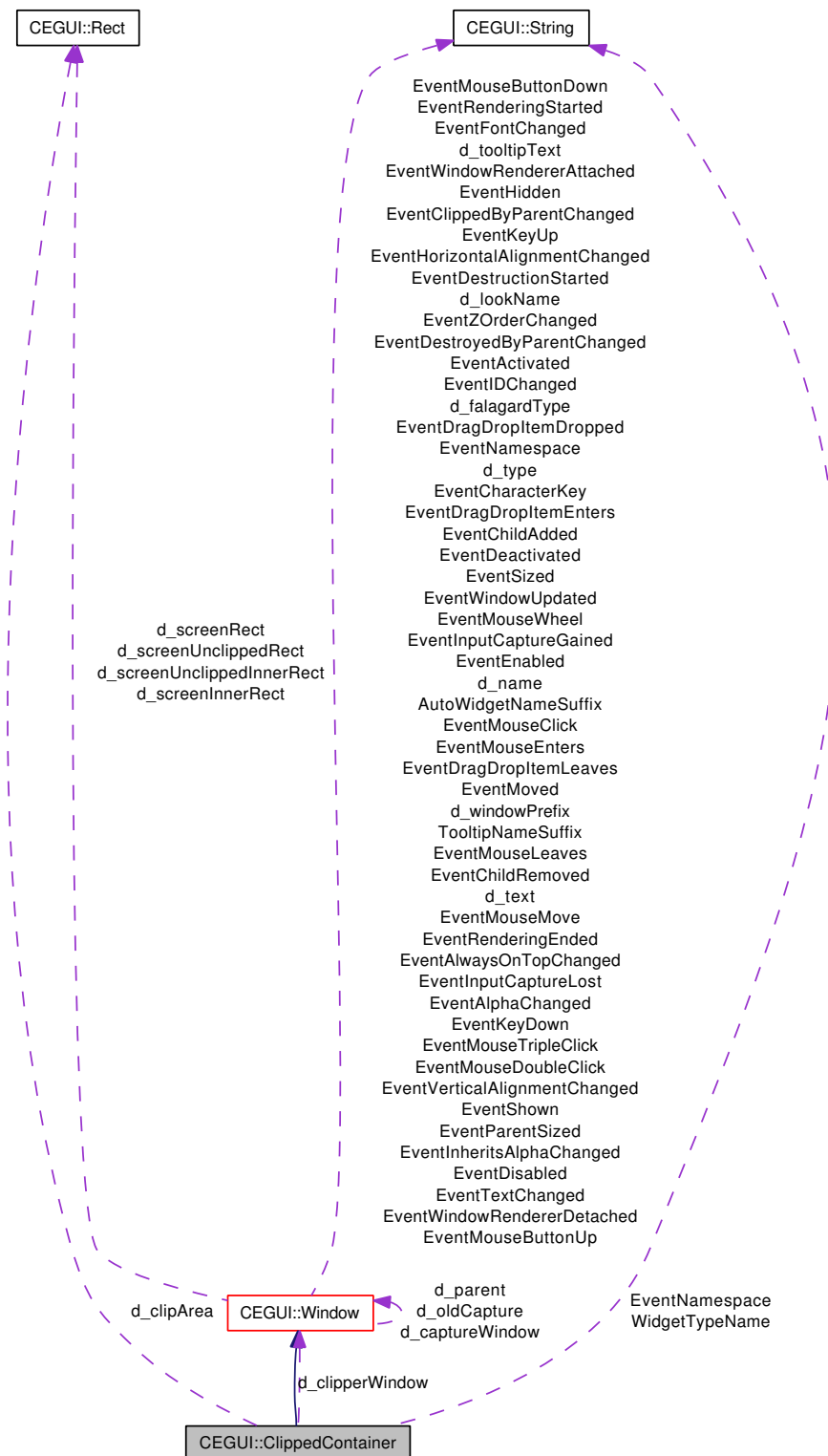
## 6.22 CEGUI::ClippedContainer Class Reference

Helper container window that has configurable clipping. Used by the [ItemListbox](#) widget.

Inheritance diagram for CEGUI::ClippedContainer:



Collaboration diagram for CEGUI::ClippedContainer:



## Public Member Functions

- [ClippedContainer](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [ClippedContainer](#) objects.*
- [~ClippedContainer](#) (void)  
*Destructor for [ClippedContainer](#) objects.*
- const [Rect](#) & [getClipArea](#) (void) const  
*Return the current clipping rectangle.*
- [Window](#) \* [getClipperWindow](#) (void) const  
*Returns the reference window used for converting the clipper rect to screen space.*
- void [setClipArea](#) (const [Rect](#) &r)  
*Set the custom clipper area in pixels.*
- void [setClipperWindow](#) ([Window](#) \*w)  
*Set the clipper reference window.*
- virtual [Rect](#) [getUnclippedInnerRect\\_impl](#) (void) const  
*Return a [Rect](#) object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.*

## Static Public Attributes

- static const [String](#) [WidgetTypeName](#)  
*Type name for [ClippedContainer](#).*
- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*

## Protected Member Functions

- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [drawSelf](#) (float)  
*Perform the actual rendering for this [Window](#).*

## Protected Attributes

- [Rect](#) [d\\_clipArea](#)  
*the pixel rect to be used for clipping relative to either a window or the screen.*

- [Window](#) \* [d\\_clipperWindow](#)

*the base window which the clipping rect is relative to.*

### 6.22.1 Detailed Description

Helper container window that has configurable clipping. Used by the [ItemListbox](#) widget.

### 6.22.2 Member Function Documentation

#### 6.22.2.1 `const Rect & CEGUI::ClippedContainer::getClipArea (void) const`

Return the current clipping rectangle.

**Returns:**

[Rect](#) object describing the clipping area in pixel that will be applied during rendering.

#### 6.22.2.2 `void CEGUI::ClippedContainer::setClipperWindow (CEGUI::Window * w)`

Set the clipper reference window.

**Parameters:**

- w* The window to be used a base for converting the custom clipper rect to screen space. NULL if the clipper rect is relative to the screen.

#### 6.22.2.3 `Rect CEGUI::ClippedContainer::getUnclippedInnerRect_impl (void) const` [virtual]

Return a [Rect](#) object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.

**Returns:**

[Rect](#) object that describes, in unclipped screen pixel co-ordinates, the window object's inner rect area.

Reimplemented from [CEGUI::Window](#).

#### 6.22.2.4 `virtual bool CEGUI::ClippedContainer::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

- class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.22.2.5 virtual void CEGUI::ClippedContainer::drawSelf (float z) [inline, protected, virtual]**

Perform the actual rendering for this [Window](#).

**Parameters:**

*z* float value specifying the base Z co-ordinate that should be used when rendering

**Returns:**

Nothing

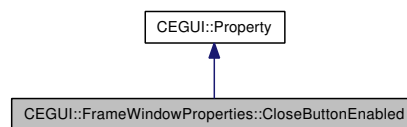
Reimplemented from [CEGUI::Window](#).



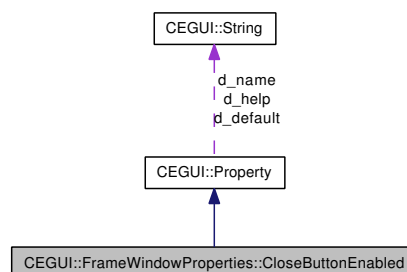
## 6.23 CEGUI::FrameWindowProperties::CloseButtonEnabled Class Reference

[Property](#) to access the setting for whether the window close button will be enabled (or displayed depending upon choice of final widget type).

Inheritance diagram for CEGUI::FrameWindowProperties::CloseButtonEnabled:



Collaboration diagram for CEGUI::FrameWindowProperties::CloseButtonEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.23.1 Detailed Description

[Property](#) to access the setting for whether the window close button will be enabled (or displayed depending upon choice of final widget type).

#### Usage:

- Name: [CloseButtonEnabled](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the windows close button should be enabled (and/or visible)
- "False" to indicate the windows close button should be disabled (and/or hidden)

## 6.23.2 Member Function Documentation

### 6.23.2.1 String CEGUI::FrameWindowProperties::CloseButtonEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.23.2.2 void CEGUI::FrameWindowProperties::CloseButtonEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.24 CEGUI::colour Class Reference

Class representing [colour](#) values within the system.

### Public Member Functions

- **colour** (const [colour](#) &val)
- **colour** (float red, float green, float blue, float alpha=1.0f)
- **colour** ([argb\\_t](#) argb)
- [argb\\_t](#) **getARGB** (void) const
- float **getAlpha** (void) const
- float **getRed** (void) const
- float **getGreen** (void) const
- float **getBlue** (void) const
- float **getHue** (void) const
- float **getSaturation** (void) const
- float **getLumination** (void) const
- void **setARGB** ([argb\\_t](#) argb)
- void **setAlpha** (float alpha)
- void **setRed** (float red)
- void **setGreen** (float green)
- void **setBlue** (float blue)
- void **set** (float red, float green, float blue, float alpha=1.0f)
- void **setRGB** (float red, float green, float blue)
- void **setRGB** (const [colour](#) &val)
- void **setHSL** (float hue, float saturation, float luminance, float alpha=1.0f)
- void **invertColour** (void)
- void **invertColourWithAlpha** (void)
- [colour](#) & **operator=** ([argb\\_t](#) val)
- [colour](#) & **operator=** (const [colour](#) &val)
- [colour](#) & **operator &=** ([argb\\_t](#) val)
- [colour](#) & **operator &=** (const [colour](#) &val)
- [colour](#) & **operator|=** ([argb\\_t](#) val)
- [colour](#) & **operator|=** (const [colour](#) &val)
- [colour](#) & **operator<<=** (int val)
- [colour](#) & **operator>>=** (int val)
- [colour](#) **operator+** (const [colour](#) &val) const
- [colour](#) **operator-** (const [colour](#) &val) const
- [colour](#) **operator \*** (const float val) const
- [colour](#) & **operator \*=** (const [colour](#) &val)
- bool **operator==** (const [colour](#) &rhs) const
- bool **operator!=** (const [colour](#) &rhs) const
- **operator argb\_t** () const

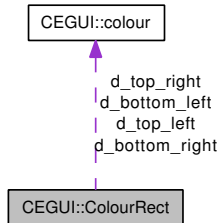
### 6.24.1 Detailed Description

Class representing [colour](#) values within the system.

## 6.25 CEGUI::ColourRect Class Reference

Class that holds details of colours for the four corners of a rectangle.

Collaboration diagram for CEGUI::ColourRect:



### Public Member Functions

- **ColourRect** (void)  
*Default constructor.*
- **ColourRect** (const **colour** &col)  
*Constructor for **ColourRect** objects (via single **colour**). Also handles default construction.*
- **ColourRect** (const **colour** &top\_left, const **colour** &top\_right, const **colour** &bottom\_left, const **colour** &bottom\_right)  
*Constructor for **ColourRect** objects.*
- void **setAlpha** (float alpha)  
*Set the alpha value to use for all four corners of the **ColourRect**.*
- void **setTopAlpha** (float alpha)  
*Set the alpha value to use for the top edge of the **ColourRect**.*
- void **setBottomAlpha** (float alpha)  
*Set the alpha value to use for the bottom edge of the **ColourRect**.*
- void **setLeftAlpha** (float alpha)  
*Set the alpha value to use for the left edge of the **ColourRect**.*
- void **setRightAlpha** (float alpha)  
*Set the alpha value to use for the right edge of the **ColourRect**.*
- bool **isMonochromatic** () const  
*Determinate the **ColourRect** is monochromatic or variegated.*
- **ColourRect** **getSubRectangle** (float left, float right, float top, float bottom) const  
*Gets a portion of this **ColourRect** as a subset **ColourRect**.*
- **colour** **getColourAtPoint** (float x, float y) const  
*Get the **colour** at a point in the rectangle.*

- void `setColours` (const `colour` &col)  
*Set the `colour` of all four corners simultaneously.*
- void `modulateAlpha` (float alpha)  
*Module the alpha components of each corner's `colour` by a constant.*
- `ColourRect` & `operator *=` (const `ColourRect` &other)  
*Modulate all components of this `colour` rect with corresponding components from another `colour` rect.*

## Public Attributes

- `colour d_top_left`
- `colour d_top_right`
- `colour d_bottom_left`
- `colour d_bottom_right`

### 6.25.1 Detailed Description

Class that holds details of colours for the four corners of a rectangle.

### 6.25.2 Member Function Documentation

#### 6.25.2.1 void CEGUI::ColourRect::setAlpha (float *alpha*)

Set the alpha value to use for all four corners of the `ColourRect`.

##### Parameters:

*alpha* Alpha value to use.

##### Returns:

Nothing.

#### 6.25.2.2 void CEGUI::ColourRect::setTopAlpha (float *alpha*)

Set the alpha value to use for the top edge of the `ColourRect`.

##### Parameters:

*alpha* Alpha value to use.

##### Returns:

Nothing.

**6.25.2.3 void CEGUI::ColourRect::setBottomAlpha (float *alpha*)**

Set the alpha value to use for the bottom edge of the [ColourRect](#).

**Parameters:**

*alpha* Alpha value to use.

**Returns:**

Nothing.

**6.25.2.4 void CEGUI::ColourRect::setLeftAlpha (float *alpha*)**

Set the alpha value to use for the left edge of the [ColourRect](#).

**Parameters:**

*alpha* Alpha value to use.

**Returns:**

Nothing.

**6.25.2.5 void CEGUI::ColourRect::setRightAlpha (float *alpha*)**

Set the alpha value to use for the right edge of the [ColourRect](#).

**Parameters:**

*alpha* Alpha value to use.

**Returns:**

Nothing.

**6.25.2.6 bool CEGUI::ColourRect::isMonochromatic () const**

Determinate the [ColourRect](#) is monochromatic or variegated.

**Returns:**

True if all four corners of the [ColourRect](#) has same [colour](#), false otherwise.

**6.25.2.7 ColourRect CEGUI::ColourRect::getSubRectangle (float *left*, float *right*, float *top*, float *bottom*) const**

Gets a portion of this [ColourRect](#) as a subset [ColourRect](#).

**Parameters:**

*left* The left side of this subrectangle (in the range of 0-1 float)

*right* The right side of this subrectangle (in the range of 0-1 float)

*top* The top side of this subrectangle (in the range of 0-1 float)

*bottom* The bottom side of this subrectangle (in the range of 0-1 float)

**Returns:**

A [ColourRect](#) from the specified range

**6.25.2.8 colour CEGUI::ColourRect::getColourAtPoint (float *x*, float *y*) const**

Get the [colour](#) at a point in the rectangle.

**Parameters:**

*x* The x coordinate of the point

*y* The y coordinate of the point

**Returns:**

The [colour](#) at the specified point.

**6.25.2.9 void CEGUI::ColourRect::setColours (const colour & *col*)**

Set the [colour](#) of all four corners simultaneously.

**Parameters:**

*col* [colour](#) that is to be set for all four corners of the [ColourRect](#);

**6.25.2.10 void CEGUI::ColourRect::modulateAlpha (float *alpha*)**

Module the alpha components of each corner's [colour](#) by a constant.

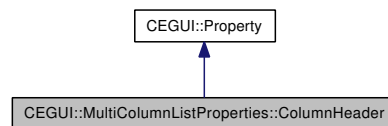
**Parameters:**

*alpha* The constant factor to modulate all alpha [colour](#) components by.

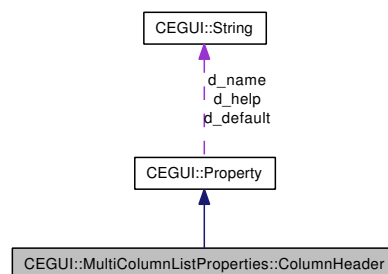
## 6.26 CEGUI::MultiColumnListProperties::ColumnHeader Class Reference

[Property](#) to access a column.

Inheritance diagram for CEGUI::MultiColumnListProperties::ColumnHeader:



Collaboration diagram for CEGUI::MultiColumnListProperties::ColumnHeader:



### Public Member Functions

- `String` `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.26.1 Detailed Description

[Property](#) to access a column.

#### Usage:

- Name: [ColumnHeader](#)
- Format: "text:[caption] width:{s,o} id:[uint]"

#### where:

- [caption] is the column header caption text.
- [{s,o}] is a [UDim](#) specification.
- [uint] is the unique ID for the column.



## 6.26.2 Member Function Documentation

### 6.26.2.1 String CEGUI::MultiColumnListProperties::ColumnHeader::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.26.2.2 void CEGUI::MultiColumnListProperties::ColumnHeader::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

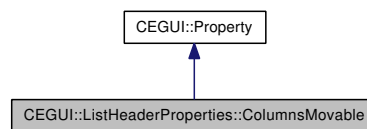
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

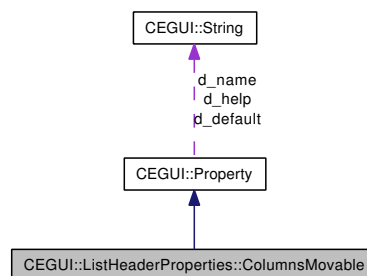
## 6.27 CEGUI::ListHeaderProperties::ColumnsMovable Class Reference

[Property](#) to access the setting for user moving of the column headers.

Inheritance diagram for CEGUI::ListHeaderProperties::ColumnsMovable:



Collaboration diagram for CEGUI::ListHeaderProperties::ColumnsMovable:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.27.1 Detailed Description

[Property](#) to access the setting for user moving of the column headers.

Usage:

- Name: [ColumnsMovable](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the column headers can be moved by the user.
- "False" to indicate the column headers can not be moved by the user.

## 6.27.2 Member Function Documentation

### 6.27.2.1 String CEGUI::ListHeaderProperties::ColumnsMovable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.27.2.2 void CEGUI::ListHeaderProperties::ColumnsMovable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

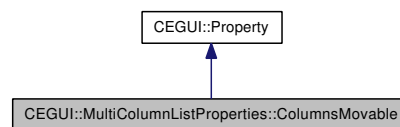
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

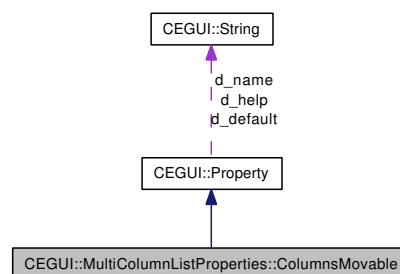
## 6.28 CEGUI::MultiColumnListProperties::ColumnsMovable Class Reference

[Property](#) to access the setting for user moving of the column headers.

Inheritance diagram for CEGUI::MultiColumnListProperties::ColumnsMovable:



Collaboration diagram for CEGUI::MultiColumnListProperties::ColumnsMovable:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.28.1 Detailed Description

[Property](#) to access the setting for user moving of the column headers.

#### Usage:

- Name: [ColumnsMovable](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate the column headers can be moved by the user.
- "False" to indicate the column headers can not be moved by the user.

## 6.28.2 Member Function Documentation

### 6.28.2.1 String CEGUI::MultiColumnListProperties::ColumnsMovable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.28.2.2 void CEGUI::MultiColumnListProperties::ColumnsMovable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

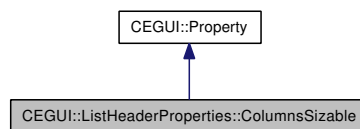
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

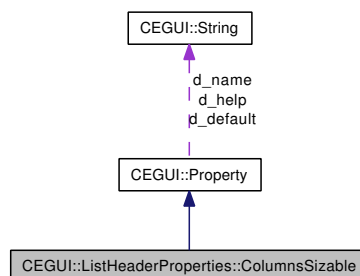
## 6.29 CEGUI::ListHeaderProperties::ColumnsSizable Class Reference

[Property](#) to access the setting for user sizing of the column headers.

Inheritance diagram for CEGUI::ListHeaderProperties::ColumnsSizable:



Collaboration diagram for CEGUI::ListHeaderProperties::ColumnsSizable:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.29.1 Detailed Description

[Property](#) to access the setting for user sizing of the column headers.

Usage:

- Name: [ColumnsSizable](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the column headers can be sized by the user.
- "False" to indicate the column headers can not be sized by the user.

## 6.29.2 Member Function Documentation

### 6.29.2.1 String CEGUI::ListHeaderProperties::ColumnsSizable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.29.2.2 void CEGUI::ListHeaderProperties::ColumnsSizable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

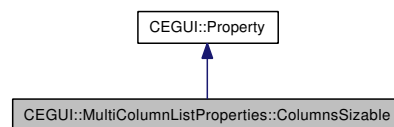
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

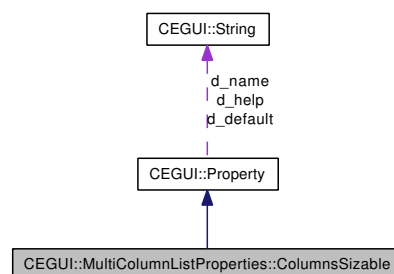
## 6.30 CEGUI::MultiColumnListProperties::ColumnsSizable Class Reference

[Property](#) to access the setting for user sizing of the column headers.

Inheritance diagram for CEGUI::MultiColumnListProperties::ColumnsSizable:



Collaboration diagram for CEGUI::MultiColumnListProperties::ColumnsSizable:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.30.1 Detailed Description

[Property](#) to access the setting for user sizing of the column headers.

#### Usage:

- Name: [ColumnsSizable](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate the column headers can be sized by the user.
- "False" to indicate the column headers can not be sized by the user.



## 6.30.2 Member Function Documentation

### 6.30.2.1 String CEGUI::MultiColumnListProperties::ColumnsSizable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.30.2.2 void CEGUI::MultiColumnListProperties::ColumnsSizable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

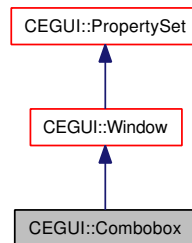
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

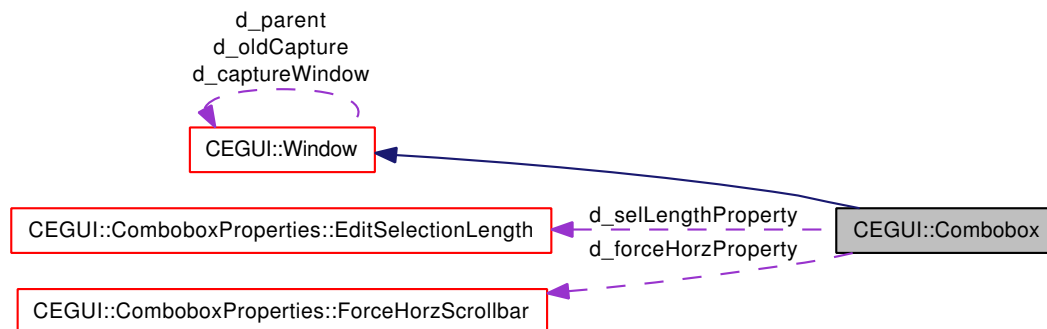
## 6.31 CEGUI::Combobox Class Reference

Base class for the [Combobox](#) widget.

Inheritance diagram for CEGUI::Combobox:



Collaboration diagram for CEGUI::Combobox:



### Public Member Functions

- virtual bool **isHit** (const [Point](#) &position) const  
*check if the given position would hit this window.*
- bool **getSingleClickEnabled** (void) const  
*returns the mode of operation for the combo box.*
- bool **isDropDownListVisible** (void) const  
*returns true if the drop down list is visible.*
- [Editbox](#) \* **getEditbox** () const  
*Return a pointer to the [Editbox](#) component widget for this [Combobox](#).*
- [PushButton](#) \* **getPushButton** () const  
*Return a pointer to the [PushButton](#) component widget for this [Combobox](#).*
- [ComboDropList](#) \* **getDropList** () const  
*Return a pointer to the [ComboDropList](#) component widget for this [Combobox](#).*
- bool **hasInputFocus** (void) const

*return true if the [Editbox](#) has input focus.*

- bool [isReadOnly](#) (void) const  
*return true if the [Editbox](#) is read-only.*
- bool [isTextValid](#) (void) const  
*return true if the [Editbox](#) text is valid given the currently set validation string.*
- const [String](#) & [getValidationString](#) (void) const  
*return the currently set validation string*
- size\_t [getCaratIndex](#) (void) const  
*return the current position of the carat.*
- size\_t [getSelectionStartIndex](#) (void) const  
*return the current selection start point.*
- size\_t [getSelectionEndIndex](#) (void) const  
*return the current selection end point.*
- size\_t [getSelectionLength](#) (void) const  
*return the length of the current selection (in code points / characters).*
- size\_t [getMaxTextLength](#) (void) const  
*return the maximum text length set for this [Editbox](#).*
- size\_t [getItemCount](#) (void) const  
*Return number of items attached to the list box.*
- [ListboxItem](#) \* [getSelectedItem](#) (void) const  
*Return a pointer to the currently selected item.*
- [ListboxItem](#) \* [getListboxItemFromIndex](#) (size\_t index) const  
*Return the item at index position index.*
- size\_t [getItemIndex](#) (const [ListboxItem](#) \*item) const  
*Return the index of [ListboxItem](#) item.*
- bool [isSortEnabled](#) (void) const  
*return whether list sorting is enabled*
- bool [isItemSelected](#) (size\_t index) const  
*return whether the string at index position index is selected*
- [ListboxItem](#) \* [findItemWithText](#) (const [String](#) &text, const [ListboxItem](#) \*start\_item)  
*Search the list for an item with the specified text.*
- bool [isListboxItemInList](#) (const [ListboxItem](#) \*item) const  
*Return whether the specified [ListboxItem](#) is in the List.*

- bool [isVertScrollbarAlwaysShown](#) (void) const  
*Return whether the vertical scroll bar is always shown.*
- bool [isHorzScrollbarAlwaysShown](#) (void) const  
*Return whether the horizontal scroll bar is always shown.*
- virtual void [initialiseComponents](#) (void)  
*Initialise the [Window](#) based object ready for use.*
- void [showDropList](#) (void)  
*Show the drop-down list.*
- void [hideDropList](#) (void)  
*Hide the drop-down list.*
- void [setSingleClickEnabled](#) (bool setting)  
*Set the mode of operation for the combo box.*
- void [setReadOnly](#) (bool setting)  
*Specify whether the [Editbox](#) is read-only.*
- void [setValidationString](#) (const [String](#) &validation\_string)  
*Set the text validation string.*
- void [setCaratIndex](#) (size\_t carat\_pos)  
*Set the current position of the carat.*
- void [setSelection](#) (size\_t start\_pos, size\_t end\_pos)  
*Define the current selection for the [Editbox](#).*
- void [setMaxTextLength](#) (size\_t max\_len)  
*set the maximum text length for this [Editbox](#).*
- void [activateEditbox](#) (void)  
*Activate the edit box component of the [Combobox](#).*
- void [resetList](#) (void)  
*Remove all items from the list.*
- void [addItem](#) ([ListboxItem](#) \*item)  
*Add the given [ListboxItem](#) to the list.*
- void [insertItem](#) ([ListboxItem](#) \*item, const [ListboxItem](#) \*position)  
*Insert an item into the list box after a specified item already in the list.*
- void [removeItem](#) (const [ListboxItem](#) \*item)  
*Removes the given item from the list box.*
- void [clearAllSelections](#) (void)  
*Clear the selected state for all items.*

- void [setSortingEnabled](#) (bool setting)  
*Set whether the list should be sorted.*
- void [setShowVertScrollbar](#) (bool setting)  
*Set whether the vertical scroll bar should always be shown.*
- void [setShowHorzScrollbar](#) (bool setting)  
*Set whether the horizontal scroll bar should always be shown.*
- void [setItemSelectState](#) ([ListboxItem](#) \*item, bool state)  
*Set the select state of an attached [ListboxItem](#).*
- void [setItemSelectState](#) (size\_t item\_index, bool state)  
*Set the select state of an attached [ListboxItem](#).*
- void [handleUpdatedListItemData](#) (void)  
*Causes the list box to update it's internal state after changes have been made to one or more attached [ListboxItem](#) objects.*
- [Combobox](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Combobox](#) base class.*
- virtual [~Combobox](#) (void)  
*Destructor for [Combobox](#) base class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventReadOnlyModeChanged](#)  
*The read-only mode for the edit box has been changed.*
- static const [String](#) [EventValidationStringChanged](#)  
*The validation string has been changed.*
- static const [String](#) [EventMaximumTextLengthChanged](#)  
*The maximum allowable string length has been changed.*
- static const [String](#) [EventTextInvalidated](#)  
*Some operation has made the current text invalid with regards to the validation string.*
- static const [String](#) [EventInvalidEntryAttempted](#)  
*The user attempted to modify the text in a way that would have made it invalid.*

- static const [String EventCaratMoved](#)  
*The text carat (insert point) has changed.*
- static const [String EventTextSelectionChanged](#)  
*The current text selection has changed.*
- static const [String EventEditboxFull](#)  
*The number of characters in the edit box has reached the current maximum.*
- static const [String EventTextAccepted](#)  
*The user has accepted the current text by pressing Return, Enter, or Tab.*
- static const [String EventListContentsChanged](#)  
*Event triggered when the contents of the list is changed.*
- static const [String EventListSelectionChanged](#)  
*Event triggered when there is a change to the currently selected item(s).*
- static const [String EventSortModeChanged](#)  
*Event triggered when the sort mode setting changes.*
- static const [String EventVertScrollbarModeChanged](#)  
*Event triggered when the vertical scroll bar 'force' setting changes.*
- static const [String EventHorzScrollbarModeChanged](#)  
*Event triggered when the horizontal scroll bar 'force' setting changes.*
- static const [String EventDropListDisplayed](#)  
*Event triggered when the drop-down list is displayed.*
- static const [String EventDropListRemoved](#)  
*Event triggered when the drop-down list is removed / hidden.*
- static const [String EventListSelectionAccepted](#)  
*Event triggered when the user accepts a selection from the drop-down list.*
- static const [String EditboxNameSuffix](#)  
*Widget name suffix for the editbox component.*
- static const [String DropListNameSuffix](#)  
*Widget name suffix for the drop list component.*
- static const [String ButtonNameSuffix](#)  
*Widget name suffix for the button component.*

## Protected Member Functions

- bool [button\\_PressHandler](#) (const [EventArgs](#) &e)  
*Handler function for button clicks.*
- bool [droplist\\_SelectionAcceptedHandler](#) (const [EventArgs](#) &e)  
*Handler for selections made in the drop-list.*
- bool [droplist\\_HiddenHandler](#) (const [EventArgs](#) &e)  
*Handler for when drop-list hides itself.*
- bool [editbox\\_MouseDownHandler](#) (const [EventArgs](#) &e)  
*Mouse button down handler attached to edit box.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- bool [editbox\\_ReadOnlyChangedHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_ValidationStringChangedHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_MaximumTextLengthChangedHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_TextInvalidatedEventHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_InvalidEntryAttemptedHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_CaratMovedHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_TextSelectionChangedHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_EditboxFullEventHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_TextAcceptedEventHandler](#) (const [EventArgs](#) &e)
- bool [editbox\\_TextChangedEventHandler](#) (const [EventArgs](#) &e)
- bool [listbox\\_ListContentsChangedHandler](#) (const [EventArgs](#) &e)
- bool [listbox\\_ListSelectionChangedHandler](#) (const [EventArgs](#) &e)
- bool [listbox\\_SortModeChangedHandler](#) (const [EventArgs](#) &e)
- bool [listbox\\_VertScrollModeChangedHandler](#) (const [EventArgs](#) &e)
- bool [listbox\\_HorzScrollModeChangedHandler](#) (const [EventArgs](#) &e)
- virtual void [onReadOnlyChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the read only state of the Combobox's [Editbox](#) has been changed.*
- virtual void [onValidationStringChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the Combobox's [Editbox](#) validation string has been changed.*
- virtual void [onMaximumTextLengthChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the Combobox's [Editbox](#) maximum text length is changed.*
- virtual void [onTextInvalidatedEvent](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the Combobox's [Editbox](#) text has been invalidated.*
- virtual void [onInvalidEntryAttempted](#) ([WindowEventArgs](#) &e)  
*Handler called internally when an invalid entry was attempted in the Combobox's [Editbox](#).*
- virtual void [onCaratMoved](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the carat in the Combobox's [Editbox](#) moves.*

- virtual void [onTextSelectionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the selection within the Combobox's [Editbox](#) changes.*
- virtual void [onEditboxFullEvent](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the maximum length is reached for text in the Combobox's [Editbox](#).*
- virtual void [onTextAcceptedEvent](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the text in the Combobox's [Editbox](#) is accepted (by various means).*
- virtual void [onListContentsChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the Combobox's Drop-down list contents are changed.*
- virtual void [onListSelectionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the selection within the Combobox's drop-down list changes (this is not the 'final' accepted selection, just the currently highlighted item).*
- virtual void [onSortModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called fired internally when the sort mode for the Combobox's drop-down list is changed.*
- virtual void [onVertScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the 'force' setting for the vertical scrollbar within the Combobox's drop-down list is changed.*
- virtual void [onHorzScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the 'force' setting for the horizontal scrollbar within the Combobox's drop-down list is changed.*
- virtual void [onDropListDisplayed](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the Combobox's drop-down list has been displayed.*
- virtual void [onDropListRemoved](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the Combobox's drop-down list has been hidden.*
- virtual void [onListSelectionAccepted](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the user has confirmed a selection within the Combobox's drop-down list.*
- virtual void [onFontChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's font is changed.*
- virtual void [onTextChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's text is changed.*
- virtual void [onActivated](#) ([ActivationEventArgs](#) &e)  
*Handler called when this window has become the active window.*

## Protected Attributes

- bool [d\\_singleClickOperation](#)  
*true if user can show and select from list in a single click.*



### 6.31.1 Detailed Description

Base class for the [Combobox](#) widget.

### 6.31.2 Member Function Documentation

#### 6.31.2.1 `virtual bool CEGUI::Combobox::isHit (const Point & position) const` [inline, virtual]

check if the given position would hit this window.

##### Parameters:

*position* Point object describing the position to check in screen pixels

##### Returns:

true if *position* 'hits' this [Window](#), else false.

Reimplemented from [CEGUI::Window](#).

#### 6.31.2.2 `bool CEGUI::Combobox::getSingleClickEnabled (void) const`

returns the mode of operation for the combo box.

##### Returns:

- true if the user can show the list and select an item with a single mouse click.
- false if the user must click to show the list and then click again to select an item.

#### 6.31.2.3 `bool CEGUI::Combobox::isDropDownListVisible (void) const`

returns true if the drop down list is visible.

##### Returns:

true if the drop down list is visible, false otherwise.

#### 6.31.2.4 `Editbox * CEGUI::Combobox::getEditbox () const`

Return a pointer to the [Editbox](#) component widget for this [Combobox](#).

##### Returns:

Pointer to an [Editbox](#) object.

##### Exceptions:

[UnknownObjectException](#) Thrown if the [Editbox](#) component does not exist.

**6.31.2.5    `PushButton *` `CEGUI::Combobox::getPushButton ()` `const`**

Return a pointer to the [PushButton](#) component widget for this [Combobox](#).

**Returns:**

Pointer to a [PushButton](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the [PushButton](#) component does not exist.

**6.31.2.6    `ComboDropList *` `CEGUI::Combobox::getDropList ()` `const`**

Return a pointer to the [ComboDropList](#) component widget for this [Combobox](#).

**Returns:**

Pointer to an [ComboDropList](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the [ComboDropList](#) component does not exist.

**6.31.2.7    `bool` `CEGUI::Combobox::hasInputFocus (void)` `const`**

return true if the [Editbox](#) has input focus.

**Returns:**

true if the [Editbox](#) has keyboard input focus, false if the [Editbox](#) does not have keyboard input focus.

**6.31.2.8    `bool` `CEGUI::Combobox::isReadOnly (void)` `const`**

return true if the [Editbox](#) is read-only.

**Returns:**

true if the [Editbox](#) is read only and can't be edited by the user, false if the [Editbox](#) is not read only and may be edited by the user.

**6.31.2.9    `bool` `CEGUI::Combobox::isTextValid (void)` `const`**

return true if the [Editbox](#) text is valid given the currently set validation string.

**Note:**

It is possible to programmatically set 'invalid' text for the [Editbox](#) by calling `setText`. This has certain implications since if invalid text is set, whatever the user types into the box will be rejected when the input is validated.

Validation is performed by means of a regular expression. If the text matches the regex, the text is said to have passed validation. If the text does not match with the regex then the text fails validation.

**Returns:**

true if the current [Editbox](#) text passes validation, false if the text does not pass validation.

**6.31.2.10 const String & CEGUI::Combobox::getValidationString (void) const**

return the currently set validation string

**Note:**

Validation is performed by means of a regular expression. If the text matches the regex, the text is said to have passed validation. If the text does not match with the regex then the text fails validation.

**Returns:**

[String](#) object containing the current validation regex data

**6.31.2.11 size\_t CEGUI::Combobox::getCaratIndex (void) const**

return the current position of the carat.

**Returns:**

Index of the insert carat relative to the start of the text.

**6.31.2.12 size\_t CEGUI::Combobox::getSelectionStartIndex (void) const**

return the current selection start point.

**Returns:**

Index of the selection start point relative to the start of the text. If no selection is defined this function returns the position of the carat.

**6.31.2.13 size\_t CEGUI::Combobox::getSelectionEndIndex (void) const**

return the current selection end point.

**Returns:**

Index of the selection end point relative to the start of the text. If no selection is defined this function returns the position of the carat.

**6.31.2.14 size\_t CEGUI::Combobox::getSelectionLength (void) const**

return the length of the current selection (in code points / characters).

**Returns:**

Number of code points (or characters) contained within the currently defined selection.

**6.31.2.15   size\_t CEGUI::Combobox::getMaxTextLength (void) const**

return the maximum text length set for this [Editbox](#).

**Returns:**

The maximum number of code points (characters) that can be entered into this [Editbox](#).

**Note:**

Depending on the validation string set, the actual length of text that can be entered may be less than the value returned here (it will never be more).

**6.31.2.16   size\_t CEGUI::Combobox::getItemCount (void) const**

Return number of items attached to the list box.

**Returns:**

the number of items currently attached to this list box.

**6.31.2.17   ListBoxItem \* CEGUI::Combobox::getSelectedItem (void) const**

Return a pointer to the currently selected item.

**Returns:**

Pointer to a [ListBoxItem](#) based object that is the selected item in the list. will return NULL if no item is selected.

**6.31.2.18   ListBoxItem \* CEGUI::Combobox::getListboxItemFromIndex (size\_t *index*) const**

Return the item at index position *index*.

**Parameters:**

*index* Zero based index of the item to be returned.

**Returns:**

Pointer to the [ListBoxItem](#) at index position *index* in the list box.

**Exceptions:**

[InvalidRequestException](#) thrown if *index* is out of range.

**6.31.2.19   size\_t CEGUI::Combobox::getItemIndex (const ListBoxItem \* *item*) const**

Return the index of [ListBoxItem](#) *item*.

**Parameters:**

*item* Pointer to a [ListBoxItem](#) whos zero based index is to be returned.

**Returns:**

Zero based index indicating the position of [ListBoxItem](#) *item* in the list box.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.31.2.20 bool CEGUI::Combobox::isSortEnabled (void) const**

return whether list sorting is enabled

**Returns:**

true if the list is sorted, false if the list is not sorted

**6.31.2.21 bool CEGUI::Combobox::isSelected (size\_t index) const**

return whether the string at index position *index* is selected

**Parameters:**

*index* Zero based index of the item to be examined.

**Returns:**

true if the item at *index* is selected, false if the item at *index* is not selected.

**Exceptions:**

[InvalidRequestException](#) thrown if *index* is out of range.

**6.31.2.22 ListBoxItem \* CEGUI::Combobox::findItemWithText (const String & text, const ListBoxItem \* start\_item)**

Search the list for an item with the specified text.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

*start\_item* [ListBoxItem](#) where the search is to begin, the search will not include *item*. If *item* is NULL, the search will begin from the first item in the list.

**Returns:**

Pointer to the first [ListBoxItem](#) in the list after *item* that has text matching *text*. If no item matches the criteria NULL is returned.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.31.2.23 bool CEGUI::Combobox::isListBoxItemInList (const ListBoxItem \* *item*) const**

Return whether the specified [ListBoxItem](#) is in the List.

**Returns:**

true if [ListBoxItem](#) *item* is in the list, false if [ListBoxItem](#) *item* is not in the list.

**6.31.2.24 bool CEGUI::Combobox::isVertScrollbarAlwaysShown (void) const**

Return whether the vertical scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.31.2.25 bool CEGUI::Combobox::isHorzScrollbarAlwaysShown (void) const**

Return whether the horizontal scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.31.2.26 void CEGUI::Combobox::initialiseComponents (void) [virtual]**

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.31.2.27 void CEGUI::Combobox::showDropList (void)**

Show the drop-down list.

**Returns:**

Nothing

**6.31.2.28 void CEGUI::Combobox::hideDropList (void)**

Hide the drop-down list.

**Returns:**

Nothing.

**6.31.2.29 void CEGUI::Combobox::setSingleClickEnabled (bool *setting*)**

Set the mode of operation for the combo box.

**Parameters:**

- setting*
- true if the user should be able to show the list and select an item with a single mouse click.
  - false if the user must click to show the list and then click again to select an item.

**Returns:**

Nothing.

**6.31.2.30 void CEGUI::Combobox::setReadOnly (bool *setting*)**

Specify whether the [Editbox](#) is read-only.

**Parameters:**

- setting* true if the [Editbox](#) is read only and can't be edited by the user, false if the [Editbox](#) is not read only and may be edited by the user.

**Returns:**

Nothing.

**6.31.2.31 void CEGUI::Combobox::setValidationString (const String & *validation\_string*)**

Set the text validation string.

**Note:**

Validation is performed by means of a regular expression. If the text matches the regex, the text is said to have passed validation. If the text does not match with the regex then the text fails validation.

**Parameters:**

- validation\_string* [String](#) object containing the validation regex data to be used.

**Returns:**

Nothing.

**6.31.2.32 void CEGUI::Combobox::setCaratIndex (size\_t *carat\_pos*)**

Set the current position of the carat.

**Parameters:**

*carat\_pos* New index for the insert carat relative to the start of the text. If the value specified is greater than the number of characters in the [Editbox](#), the carat is positioned at the end of the text.

**Returns:**

Nothing.

**6.31.2.33 void CEGUI::Combobox::setSelection (size\_t *start\_pos*, size\_t *end\_pos*)**

Define the current selection for the [Editbox](#).

**Parameters:**

*start\_pos* Index of the starting point for the selection. If this value is greater than the number of characters in the [Editbox](#), the selection start will be set to the end of the text.

*end\_pos* Index of the ending point for the selection. If this value is greater than the number of characters in the [Editbox](#), the selection start will be set to the end of the text.

**Returns:**

Nothing.

**6.31.2.34 void CEGUI::Combobox::setMaxTextLength (size\_t *max\_len*)**

set the maximum text length for this [Editbox](#).

**Parameters:**

*max\_len* The maximum number of code points (characters) that can be entered into this [Editbox](#).

**Note:**

Depending on the validation string set, the actual length of text that can be entered may be less than the value set here (it will never be more).

**Returns:**

Nothing.

**6.31.2.35 void CEGUI::Combobox::activateEditbox (void)**

Activate the edit box component of the [Combobox](#).

**Returns:**

Nothing.



**6.31.2.36 void CEGUI::Combobox::resetList (void)**

Remove all items from the list.

Note that this will cause 'AutoDelete' items to be deleted.

**6.31.2.37 void CEGUI::Combobox::addItem (ListboxItem \* *item*)**

Add the given [ListboxItem](#) to the list.

**Parameters:**

*item* Pointer to the [ListboxItem](#) to be added to the list. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

**Returns:**

Nothing.

**6.31.2.38 void CEGUI::Combobox::insertItem (ListboxItem \* *item*, const ListboxItem \* *position*)**

Insert an item into the list box after a specified item already in the list.

Note that if the list is sorted, the item may not end up in the requested position.

**Parameters:**

*item* Pointer to the [ListboxItem](#) to be inserted. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

*position* Pointer to a [ListboxItem](#) that *item* is to be inserted after. If this parameter is NULL, the item is inserted at the start of the list.

**Returns:**

Nothing.

**6.31.2.39 void CEGUI::Combobox::removeItem (const ListboxItem \* *item*)**

Removes the given item from the list box.

**Parameters:**

*item* Pointer to the [ListboxItem](#) that is to be removed. If *item* is not attached to this list box then nothing will happen.

**Returns:**

Nothing.

**6.31.2.40 void CEGUI::Combobox::clearAllSelections (void)**

Clear the selected state for all items.

**Returns:**

Nothing.

**6.31.2.41 void CEGUI::Combobox::setSortingEnabled (bool *setting*)**

Set whether the list should be sorted.

**Parameters:**

*setting* true if the list should be sorted, false if the list should not be sorted.

**Returns:**

Nothing.

**6.31.2.42 void CEGUI::Combobox::setShowVertScrollbar (bool *setting*)**

Set whether the vertical scroll bar should always be shown.

**Parameters:**

*setting* true if the vertical scroll bar should be shown even when it is not required. false if the vertical scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.31.2.43 void CEGUI::Combobox::setShowHorzScrollbar (bool *setting*)**

Set whether the horizontal scroll bar should always be shown.

**Parameters:**

*setting* true if the horizontal scroll bar should be shown even when it is not required. false if the horizontal scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.31.2.44 void CEGUI::Combobox::setItemSelectState (ListboxItem \* *item*, bool *state*)**

Set the select state of an attached [ListboxItem](#).

This is the recommended way of selecting and deselecting items attached to a list box as it respects the multi-select mode setting. It is possible to modify the setting on ListboxItems directly, but that approach does not respect the settings of the list box.

**Parameters:**

*item* The [ListboxItem](#) to be affected. This item must be attached to the list box.

*state* true to select the item, false to de-select the item.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *item* is not attached to this list box.

**6.31.2.45 void CEGUI::Combobox::setItemSelectState (size\_t item\_index, bool state)**

Set the select state of an attached [ListboxItem](#).

This is the recommended way of selecting and deselecting items attached to a list box as it respects the multi-select mode setting. It is possible to modify the setting on ListboxItems directly, but that approach does not respect the settings of the list box.

**Parameters:**

*item\_index* The zero based index of the [ListboxItem](#) to be affected. This must be a valid index ( $0 \leq \text{index} < \text{getItemCount}()$ )

*state* true to select the item, false to de-select the item.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *item\_index* is out of range for the list box

**6.31.2.46 void CEGUI::Combobox::handleUpdatedListItemData (void)**

Causes the list box to update it's internal state after changes have been made to one or more attached [ListboxItem](#) objects.

Client code must call this whenever it has made any changes to [ListboxItem](#) objects already attached to the list box. If you are just adding items, or removed items to update them prior to re-adding them, there is no need to call this method.

**Returns:**

Nothing.

**6.31.2.47 virtual bool CEGUI::Combobox::testClassName\_impl (const String & class\_name)  
const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.31.2.48** `void CEGUI::Combobox::onFontChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's font is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.31.2.49** `void CEGUI::Combobox::onTextChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.31.2.50** `void CEGUI::Combobox::onActivated (ActivationEventArgs & e)` [protected, virtual]

Handler called when this window has become the active window.

**Parameters:**

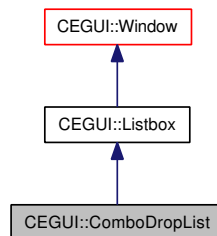
- e* [ActivationEventArgs](#) class whose 'otherWindow' field is set to the window that previously was active, or NULL for none.

Reimplemented from [CEGUI::Window](#).

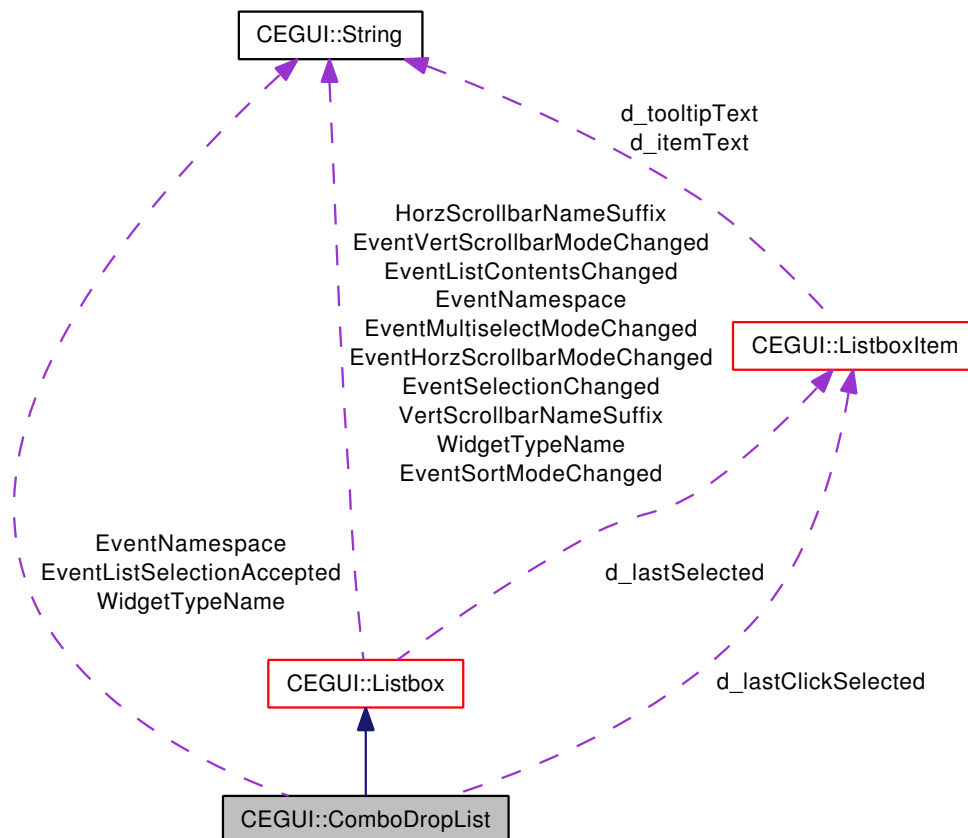
## 6.32 CEGUI::ComboDropList Class Reference

Base class for the combo box drop down list. This is a specialisation of the [Listbox](#) class.

Inheritance diagram for CEGUI::ComboDropList:



Collaboration diagram for CEGUI::ComboDropList:



### Public Member Functions

- virtual void [initialiseComponents](#) (void)  
*Initialise the [Window](#) based object ready for use.*
- void [setArmed](#) (bool setting)

*Set whether the drop-list is 'armed' for selection.*

- bool `isArmed` (void) const  
*Return the 'armed' state of the `ComboDropList`.*
- void `setAutoArmEnabled` (bool setting)  
*Set the mode of operation for the `ComboDropList`.*
- bool `isAutoArmEnabled` (void) const  
*returns the mode of operation for the drop-list*
- `ComboDropList` (const `String` &type, const `String` &name)  
*Constructor for `ComboDropList` base class.*
- virtual `~ComboDropList` (void)  
*Destructor for `ComboDropList` base class.*

## Static Public Attributes

- static const `String` `EventNamespace`  
*Namespace for global events.*
- static const `String` `WidgetTypeName`  
*`Window` factory name.*
- static const `String` `EventListSelectionAccepted`  
*`Event` fired when the user confirms the selection by clicking the mouse.*

## Protected Member Functions

- virtual bool `testClassName_impl` (const `String` &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- void `onListSelectionAccepted` (`WindowEventArgs` &e)  
*Handler for when list selection is confirmed.*
- virtual void `onMouseMove` (`MouseEventArgs` &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void `onMouseButtonDown` (`MouseEventArgs` &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void `onMouseButtonUp` (`MouseEventArgs` &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void `onCaptureLost` (`WindowEventArgs` &e)

*Handler called when this window loses capture of mouse inputs.*

- virtual void [onActivated](#) ([ActivationEventArgs](#) &e)  
*Handler called when this window has become the active window.*
- virtual void [onListContentsChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the list contents are changed.*
- virtual void [onSelectionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the currently selected item or items changes.*

## Protected Attributes

- bool [d\\_autoArm](#)  
*true if the box auto-arms when the mouse enters it.*
- bool [d\\_armed](#)  
*true when item selection has been armed.*
- [ListboxItem](#) \* [d\\_lastClickSelected](#)  
*Item last accepted by user.*

### 6.32.1 Detailed Description

Base class for the combo box drop down list. This is a specialisation of the [Listbox](#) class.

### 6.32.2 Member Function Documentation

#### 6.32.2.1 void CEGUI::ComboDropList::initialiseComponents (void) [virtual]

Initialise the [Window](#) based object ready for use.

#### Note:

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

#### Returns:

Nothing

Reimplemented from [CEGUI::Listbox](#).

#### 6.32.2.2 void CEGUI::ComboDropList::setArmed (bool *setting*) [inline]

Set whether the drop-list is 'armed' for selection.

**Note:**

This setting is not exclusively under client control; the [ComboDropList](#) will auto-arm in response to certain left mouse button events. This is also dependant upon the autoArm setting of the [ComboDropList](#).

**Parameters:**

- setting*
- true to arm the box; items will be highlighted and the next left button up event will cause dismissal and possible item selection.
  - false to disarm the box; items will not be highlighted or selected until the box is armed.

**Returns:**

Nothing.

**6.32.2.3 bool CEGUI::ComboDropList::isArmed (void) const** `[inline]`

Return the 'armed' state of the [ComboDropList](#).

**Returns:**

- true if the box is armed; items will be highlighted and the next left button up event will cause dismissal and possible item selection.
- false if the box is not armed; items will not be highlighted or selected until the box is armed.

**6.32.2.4 void CEGUI::ComboDropList::setAutoArmEnabled (bool *setting*)** `[inline]`

Set the mode of operation for the [ComboDropList](#).

**Parameters:**

- setting*
- true if the [ComboDropList](#) auto-arms when the mouse enters the box.
  - false if the user must click to arm the box.

**Returns:**

Nothing.

**6.32.2.5 bool CEGUI::ComboDropList::isAutoArmEnabled (void) const** `[inline]`

returns the mode of operation for the drop-list

**Returns:**

- true if the [ComboDropList](#) auto-arms when the mouse enters the box.
- false if the user must click to arm the box.



**6.32.2.6** `virtual bool CEGUI::ComboDropList::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Listbox](#).

**6.32.2.7** `void CEGUI::ComboDropList::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Listbox](#).

**6.32.2.8** `void CEGUI::ComboDropList::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Listbox](#).

**6.32.2.9** `void CEGUI::ComboDropList::onMouseButtonUp (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.32.2.10 void CEGUI::ComboDropList::onCaptureLost (WindowEventArgs & e)**  
[protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.32.2.11 void CEGUI::ComboDropList::onActivated (ActivationEventArgs & e)**  
[protected, virtual]

Handler called when this window has become the active window.

**Parameters:**

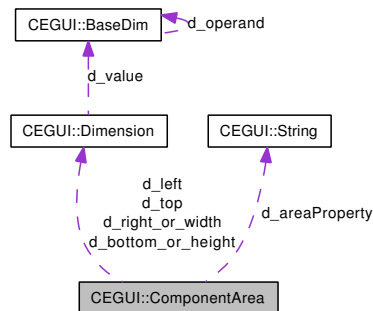
- e* [ActivationEventArgs](#) class whose 'otherWindow' field is set to the window that previously was active, or NULL for none.

Reimplemented from [CEGUI::Window](#).

## 6.33 CEGUI::ComponentArea Class Reference

Class that represents a target area for a widget or imagery component.

Collaboration diagram for CEGUI::ComponentArea:



### Public Member Functions

- **Rect** `getPixelRect` (const **Window** &wnd) const  
Return a **Rect** describing the absolute pixel area represented by this **ComponentArea**.
- **Rect** `getPixelRect` (const **Window** &wnd, const **Rect** &container) const  
Return a **Rect** describing the absolute pixel area represented by this **ComponentArea**.
- void `writeXMLToStream` (**XMLSerializer** &xml\_stream) const  
Writes an xml representation of this **ComponentArea** to out\_stream.
- bool `isAreaFetchedFromProperty` () const  
Return whether this **ComponentArea** fetches it's area via a property on the target window.
- const **String** & `getAreaPropertySource` () const  
Return the name of the property that will be used to determine the pixel area for this **ComponentArea**.
- void `setAreaPropertySource` (const **String** &property)  
Set the name of the property that will be used to determine the pixel area for this **ComponentArea**.

### Public Attributes

- **Dimension** `d_left`  
Left edge of the area.
- **Dimension** `d_top`  
Top edge of the area.
- **Dimension** `d_right_or_width`  
Either the right edge or the width of the area.

- [Dimension d\\_bottom\\_or\\_height](#)

*Either the bottom edge or the height of the area.*

### 6.33.1 Detailed Description

Class that represents a target area for a widget or imagery component.

This is essentially a [Rect](#) built out of [Dimension](#) objects. Of note is that what would normally be the 'right' and 'bottom' edges may alternatively represent width and height depending upon what the assigned [Dimension\(s\)](#) represent.

### 6.33.2 Member Function Documentation

#### 6.33.2.1 `Rect CEGUI::ComponentArea::getPixelRect (const Window & wnd) const`

Return a [Rect](#) describing the absolute pixel area represented by this [ComponentArea](#).

##### Parameters:

*wnd* [Window](#) object to be used when calculating final pixel area.

##### Returns:

[Rect](#) object describing the pixels area represented by this [ComponentArea](#) when using *wnd* as a reference for calculating the final pixel dimensions.

#### 6.33.2.2 `Rect CEGUI::ComponentArea::getPixelRect (const Window & wnd, const Rect & container) const`

Return a [Rect](#) describing the absolute pixel area represented by this [ComponentArea](#).

##### Parameters:

*wnd* [Window](#) object to be used when calculating final pixel area.

*container* [Rect](#) object to be used as a base or container when converting relative dimensions.

##### Returns:

[Rect](#) object describing the pixels area represented by this [ComponentArea](#) when using *wnd* and *container* as a reference for calculating the final pixel dimensions.

#### 6.33.2.3 `void CEGUI::ComponentArea::writeXMLToStream (XMLSerializer & xml_stream) const`

Writes an xml representation of this [ComponentArea](#) to *out\_stream*.

##### Parameters:

*xml\_stream* Stream where xml data should be output.

##### Returns:

Nothing.

#### 6.33.2.4 bool CEGUI::ComponentArea::isAreaFetchedFromProperty () const

Return whether this [ComponentArea](#) fetches it's area via a property on the target window.

**Returns:**

- true if the area comes via a Property.
- false if the area is defined explicitly via the [Dimension](#) fields.

#### 6.33.2.5 const String & CEGUI::ComponentArea::getAreaPropertySource () const

Return the name of the property that will be used to determine the pixel area for this [ComponentArea](#).

**Returns:**

[String](#) object holding the name of a Property.

#### 6.33.2.6 void CEGUI::ComponentArea::setAreaPropertySource (const String & *property*)

Set the name of the property that will be used to determine the pixel area for this [ComponentArea](#).

**Parameters:**

*property* [String](#) object holding the name of a Property. The property should access a [URect](#) type property.

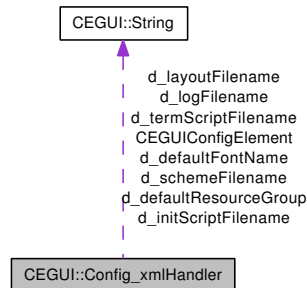
**Returns:**

Nothing.

## 6.34 CEGUI::Config\_xmlHandler Class Reference

Handler class used to parse the Configuration XML file.

Collaboration diagram for CEGUI::Config\_xmlHandler:



### Public Member Functions

- **Config\_xmlHandler** (void)  
*Constructor for [GUILayout\\_xmlHandler](#) objects.*
- virtual **~Config\_xmlHandler** (void)  
*Destructor for [GUILayout\\_xmlHandler](#) objects.*
- virtual void **elementStart** (const [String](#) &element, const [XMLAttributes](#) &attributes)  
*document processing (only care about elements, schema validates format)*
- const [String](#) & **getLogFilename** (void) const  
*Return log filename.*
- const [String](#) & **getSchemeFilename** (void) const  
*Return initial scheme filename to load.*
- const [String](#) & **getLayoutFilename** (void) const  
*Return initial layout filename to load and set as the GUI sheet.*
- const [String](#) & **getInitScriptFilename** (void) const  
*Return the name of the initialisation script to run.*
- const [String](#) & **getTermScriptFilename** (void) const  
*Return the name of the termination script to run.*
- const [String](#) & **getDefaultFontName** (void) const  
*Return name of font to use as default.*
- const [String](#) & **getDefaultResourceGroup** (void) const  
*Return name of default resource group.*
- [LogLevel](#) **getLoggingLevel** (void) const  
*Return logging level which was read from the config file.*

### 6.34.1 Detailed Description

Handler class used to parse the Configuration XML file.

## 6.35 CEGUI::ConstBaseIterator< T > Class Template Reference

Base class constant iterator used to offer iteration over various collections within the system.

### Public Types

- typedef T::mapped\_type **mapped\_type**

### Public Member Functions

- [ConstBaseIterator](#) (typename T::const\_iterator start\_iter, typename T::const\_iterator end\_iter)  
*ConstBaseIterator constructor.*
- [~ConstBaseIterator](#) (void)  
*ConstBaseIterator destructor.*
- [ConstBaseIterator](#) (const [ConstBaseIterator](#)< T > &org)  
*ConstBaseIterator copy constructor.*
- [ConstBaseIterator](#)< T > & [operator=](#) (const [ConstBaseIterator](#)< T > &rhs)  
*ConstBaseIterator assignment operator.*
- T::key\_type [getCurrentKey](#) (void) const  
*Return the key for the item at the current iterator position.*
- mapped\_type [getCurrentValue](#) (void) const  
*Return the value for the item at the current iterator position.*
- bool [isAtEnd](#) (void) const  
*Return whether the current iterator position is at the end of the iterators range.*
- bool [isAtStart](#) (void) const  
*Return whether the current iterator position is at the start of the iterators range.*
- [ConstBaseIterator](#)< T > & [operator++](#) ()  
*Increase the iterator position (prefix increment).*
- [ConstBaseIterator](#)< T > [operator++](#) (int)  
*Increase the iterator position (postfix increment).*
- [ConstBaseIterator](#)< T > & [operator--](#) ()  
*Decrease the iterator position (prefix decrement).*
- [ConstBaseIterator](#)< T > [operator--](#) (int)  
*Decrease the iterator position (postfix decrement).*
- bool [operator==](#) (const [ConstBaseIterator](#)< T > &rhs) const  
*Compares two iterators. Return true if the current position of both iterators are equivalent.*



- `bool operator!= (const ConstBaseIterator< T > &rhs) const`  
*Compares two iterators. Return true if the current position of the iterators are different.*
- `mapped_type operator* () const`  
*Return the value for the current iterator position.*
- `void toStart (void)`  
*Set the iterator current position to the start position.*
- `void toEnd (void)`  
*Set the iterator current position to the end position.*

### 6.35.1 Detailed Description

`template<class T> class CEGUI::ConstBaseIterator< T >`

Base class constant iterator used to offer iteration over various collections within the system.

### 6.35.2 Constructor & Destructor Documentation

**6.35.2.1** `template<class T> CEGUI::ConstBaseIterator< T >::ConstBaseIterator (typename T::const_iterator start_iter, typename T::const_iterator end_iter)` `[inline]`

`ConstBaseIterator` constructor.

#### Parameters:

*start\_iter* 'real' iterator that will be the start of the range to be iterated over by this iterator.

*end\_iter* 'real' iterator that will be the end of the range to be iterated over by this iterator.

### 6.35.3 Member Function Documentation

**6.35.3.1** `template<class T> ConstBaseIterator<T>& CEGUI::ConstBaseIterator< T >::operator++ ()` `[inline]`

Increase the iterator position (prefix increment).

#### Note:

The iterator is checked, and this call will always succeed, so do not rely on some exception to exit a loop.

**6.35.3.2** `template<class T> ConstBaseIterator<T> CEGUI::ConstBaseIterator< T >::operator++ (int)` `[inline]`

Increase the iterator position (postfix increment).

**Note:**

The iterator is checked, and this call will always succeed, so do not rely on some exception to exit a loop.

```
6.35.3.3 template<class T> ConstBaseIterator<T>& CEGUI::ConstBaseIterator< T
>::operator-() [inline]
```

Decrease the iterator position (prefix decrement).

**Note:**

The iterator is checked, and this call will always succeed, so do not rely on some exception to exit a loop.

```
6.35.3.4 template<class T> ConstBaseIterator<T> CEGUI::ConstBaseIterator< T >::operator-
(int) [inline]
```

Decrease the iterator position (postfix decrement).

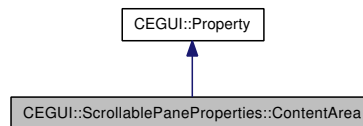
**Note:**

The iterator is checked, and this call will always succeed, so do not rely on some exception to exit a loop.

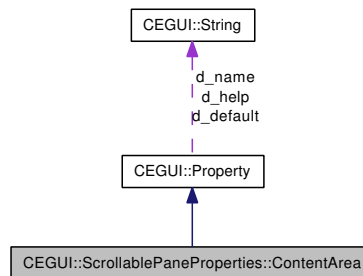
## 6.36 CEGUI::ScrollablePaneProperties::ContentArea Class Reference

[Property](#) to access the current content pane area rectangle (as window relative pixels).

Inheritance diagram for CEGUI::ScrollablePaneProperties::ContentArea:



Collaboration diagram for CEGUI::ScrollablePaneProperties::ContentArea:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.36.1 Detailed Description

[Property](#) to access the current content pane area rectangle (as window relative pixels).

**Usage:**

- Name: [ContentArea](#)
- Format: "l:[float] t:[float] r:[float] b:[float]".

**Where:**

- l:[float] specifies the position of the left edge of the area as a floating point number.
- t:[float] specifies the position of the top edge of the area as a floating point number.
- r:[float] specifies the position of the right edge of the area as a floating point number.
- b:[float] specifies the position of the bottom edge of the area as a floating point number.

## 6.36.2 Member Function Documentation

### 6.36.2.1 String CEGUI::ScrollablePaneProperties::ContentArea::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.36.2.2 void CEGUI::ScrollablePaneProperties::ContentArea::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

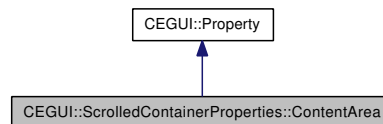
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

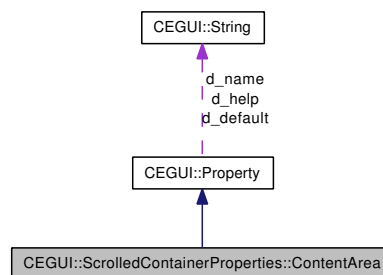
## 6.37 CEGUI::ScrolledContainerProperties::ContentArea Class Reference

[Property](#) to access the current content pane area rectangle (as window relative pixels).

Inheritance diagram for CEGUI::ScrolledContainerProperties::ContentArea:



Collaboration diagram for CEGUI::ScrolledContainerProperties::ContentArea:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.37.1 Detailed Description

[Property](#) to access the current content pane area rectangle (as window relative pixels).

**Usage:**

- Name: [ContentArea](#)
- Format: "l:[float] t:[float] r:[float] b:[float]".

**Where:**

- l:[float] specifies the position of the left edge of the area as a floating point number.
- t:[float] specifies the position of the top edge of the area as a floating point number.
- r:[float] specifies the position of the right edge of the area as a floating point number.
- b:[float] specifies the position of the bottom edge of the area as a floating point number.

## 6.37.2 Member Function Documentation

### 6.37.2.1 `String CEGUI::ScrolledContainerProperties::ContentArea::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.37.2.2 `void CEGUI::ScrolledContainerProperties::ContentArea::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

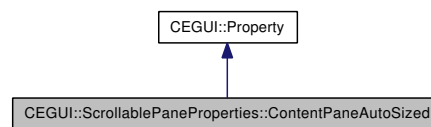
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

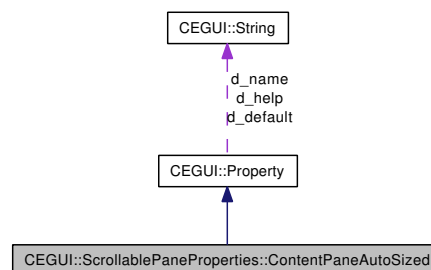
## 6.38 CEGUI::ScrollablePaneProperties::ContentPaneAutoSized Class Reference

[Property](#) to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content.

Inheritance diagram for CEGUI::ScrollablePaneProperties::ContentPaneAutoSized:



Collaboration diagram for CEGUI::ScrollablePaneProperties::ContentPaneAutoSized:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.38.1 Detailed Description

[Property](#) to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content.

Usage:

- Name: [ContentPaneAutoSized](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the pane should automatically resize itself.
- "False" to indicate the pane should not automatically resize itself.

## 6.38.2 Member Function Documentation

### 6.38.2.1 `String CEGUI::ScrollablePaneProperties::ContentPaneAutoSized::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.38.2.2 `void CEGUI::ScrollablePaneProperties::ContentPaneAutoSized::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

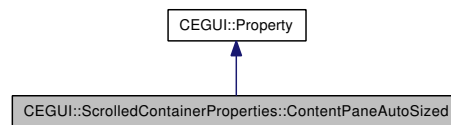
Implements [CEGUI::Property](#).



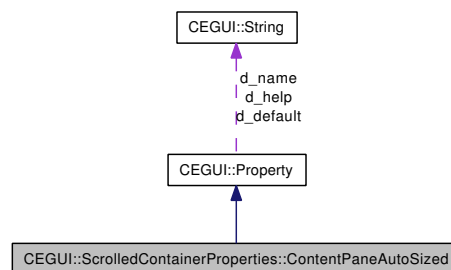
## 6.39 CEGUI::ScrolledContainerProperties::ContentPaneAutoSized Class Reference

[Property](#) to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content.

Inheritance diagram for CEGUI::ScrolledContainerProperties::ContentPaneAutoSized:



Collaboration diagram for CEGUI::ScrolledContainerProperties::ContentPaneAutoSized:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.39.1 Detailed Description

[Property](#) to access the setting which controls whether the content pane is automatically resized according to the size and position of attached content.

#### Usage:

- Name: [ContentPaneAutoSized](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate the pane should automatically resize itself.
- "False" to indicate the pane should not automatically resize itself.

## 6.39.2 Member Function Documentation

### 6.39.2.1 String CEGUI::ScrolledContainerProperties::ContentPaneAutoSized::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.39.2.2 void CEGUI::ScrolledContainerProperties::ContentPaneAutoSized::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.40 CEGUI::CoordConverter Class Reference

Utility class that helps in converting various types of co-ordinate between absolute screen positions and positions offset from the top-left corner of a given [Window](#) object.

### Static Public Member Functions

- static float [windowToScreenX](#) (const [Window](#) &window, const [UDim](#) &x)  
*Convert a window co-ordinate value, specified as a [UDim](#), to a screen relative pixel co-ordinate.*
- static float [windowToScreenX](#) (const [Window](#) &window, const float x)  
*Convert a window pixel co-ordinate value, specified as a float, to a screen pixel co-ordinate.*
- static float [windowToScreenY](#) (const [Window](#) &window, const [UDim](#) &y)  
*Convert a window co-ordinate value, specified as a [UDim](#), to a screen relative pixel co-ordinate.*
- static float [windowToScreenY](#) (const [Window](#) &window, const float y)  
*Convert a window pixel co-ordinate value, specified as a float, to a screen pixel co-ordinate.*
- static [Vector2](#) [windowToScreen](#) (const [Window](#) &window, const [UVector2](#) &vec)  
*Convert a window co-ordinate point, specified as a [UVector2](#), to a screen relative pixel co-ordinate point.*
- static [Vector2](#) [windowToScreen](#) (const [Window](#) &window, const [Vector2](#) &vec)  
*Convert a window pixel co-ordinate point, specified as a [Vector2](#), to a screen pixel co-ordinate point.*
- static [Rect](#) [windowToScreen](#) (const [Window](#) &window, const [URect](#) &rect)  
*Convert a window area, specified as a [URect](#), to a screen area.*
- static [Rect](#) [windowToScreen](#) (const [Window](#) &window, const [Rect](#) &rect)  
*Convert a pixel window area, specified as a [Rect](#), to a screen area.*
- static float [screenToWindowX](#) (const [Window](#) &window, const [UDim](#) &x)  
*Convert a screen relative [UDim](#) co-ordinate value to a window co-ordinate value, specified in pixels.*
- static float [screenToWindowX](#) (const [Window](#) &window, const float x)  
*Convert a screen pixel co-ordinate value to a window co-ordinate value, specified in pixels.*
- static float [screenToWindowY](#) (const [Window](#) &window, const [UDim](#) &y)  
*Convert a screen relative [UDim](#) co-ordinate value to a window co-ordinate value, specified in pixels.*
- static float [screenToWindowY](#) (const [Window](#) &window, const float y)  
*Convert a screen pixel co-ordinate value to a window co-ordinate value, specified in pixels.*
- static [Vector2](#) [screenToWindow](#) (const [Window](#) &window, const [UVector2](#) &vec)  
*Convert a screen relative [UVector2](#) point to a window co-ordinate point, specified in pixels.*
- static [Vector2](#) [screenToWindow](#) (const [Window](#) &window, const [Vector2](#) &vec)  
*Convert a screen [Vector2](#) pixel point to a window co-ordinate point, specified in pixels.*

- static [Rect](#) [screenToWindow](#) (const [Window](#) &window, const [URect](#) &rect)  
*Convert a [URect](#) screen area to a window area, specified in pixels.*
- static [Rect](#) [screenToWindow](#) (const [Window](#) &window, const [Rect](#) &rect)  
*Convert a [Rect](#) screen pixel area to a window area, specified in pixels.*

### 6.40.1 Detailed Description

Utility class that helps in converting various types of co-ordinate between absolute screen positions and positions offset from the top-left corner of a given [Window](#) object.

### 6.40.2 Member Function Documentation

#### 6.40.2.1 float CEGUI::CoordConverter::windowToScreenX (const [Window](#) & *window*, const [UDim](#) & *x*) [static]

Convert a window co-ordinate value, specified as a [UDim](#), to a screen relative pixel co-ordinate.

##### Parameters:

- window* [Window](#) object to use as a base for the conversion.
- x* [UDim](#) x co-ordinate value to be converted

##### Returns:

float value describing a pixel screen co-ordinate that is equivalent to window [UDim](#) co-ordinate *x*.

#### 6.40.2.2 float CEGUI::CoordConverter::windowToScreenX (const [Window](#) & *window*, const float *x*) [static]

Convert a window pixel co-ordinate value, specified as a float, to a screen pixel co-ordinate.

##### Parameters:

- window* [Window](#) object to use as a base for the conversion.
- x* float x co-ordinate value to be converted.

##### Returns:

float value describing a pixel screen co-ordinate that is equivalent to window co-ordinate *x*.

#### 6.40.2.3 float CEGUI::CoordConverter::windowToScreenY (const [Window](#) & *window*, const [UDim](#) & *y*) [static]

Convert a window co-ordinate value, specified as a [UDim](#), to a screen relative pixel co-ordinate.

##### Parameters:

- window* [Window](#) object to use as a base for the conversion.

y [UDim](#) y co-ordinate value to be converted

**Returns:**

float value describing a screen co-ordinate that is equivalent to window [UDim](#) co-ordinate y.

**6.40.2.4 float CEGUI::CoordConverter::windowToScreenY (const Window & window, const float y) [static]**

Convert a window pixel co-ordinate value, specified as a float, to a screen pixel co-ordinate.

**Parameters:**

*window* [Window](#) object to use as a base for the conversion.

y float y co-ordinate value to be converted.

**Returns:**

float value describing a screen co-ordinate that is equivalent to window co-ordinate y.

**6.40.2.5 Vector2 CEGUI::CoordConverter::windowToScreen (const Window & window, const UVector2 & vec) [static]**

Convert a window co-ordinate point, specified as a [UVector2](#), to a screen relative pixel co-ordinate point.

**Parameters:**

*window* [Window](#) object to use as a base for the conversion.

*vec* [UVector2](#) object describing the point to be converted

**Returns:**

[Vector2](#) object describing a screen co-ordinate position that is equivalent to window based [UVector2](#) *vec*.

**6.40.2.6 Vector2 CEGUI::CoordConverter::windowToScreen (const Window & window, const Vector2 & vec) [static]**

Convert a window pixel co-ordinate point, specified as a [Vector2](#), to a screen pixel co-ordinate point.

**Parameters:**

*window* [Window](#) object to use as a base for the conversion.

*vec* [Vector2](#) object describing the point to be converted.

**Returns:**

[Vector2](#) object describing a screen co-ordinate position that is equivalent to window based [Vector2](#) *vec*.

**6.40.2.7 Rect CEGUI::CoordConverter::windowToScreen (const Window & *window*, const URect & *rect*) [static]**

Convert a window area, specified as a [URect](#), to a screen area.

**Parameters:**

*rect* [URect](#) object describing the area to be converted

**Returns:**

[Rect](#) object describing a screen area that is equivalent to window area *rect*.

**6.40.2.8 Rect CEGUI::CoordConverter::windowToScreen (const Window & *window*, const Rect & *rect*) [static]**

Convert a pixel window area, specified as a [Rect](#), to a screen area.

**Parameters:**

*window* [Window](#) object to use as a base for the conversion.

*rect* [Rect](#) object describing the area to be converted.

**Returns:**

[Rect](#) object describing a screen area that is equivalent to window area *rect*.

**6.40.2.9 float CEGUI::CoordConverter::screenToWindowX (const Window & *window*, const UDim & *x*) [static]**

Convert a screen relative [UDim](#) co-ordinate value to a window co-ordinate value, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*x* [UDim](#) x co-ordinate value to be converted

**Returns:**

float value describing a window co-ordinate value that is equivalent to screen [UDim](#) co-ordinate *x*.

**6.40.2.10 float CEGUI::CoordConverter::screenToWindowX (const Window & *window*, const float *x*) [static]**

Convert a screen pixel co-ordinate value to a window co-ordinate value, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*x* float x co-ordinate value to be converted.

**Returns:**

float value describing a window co-ordinate value that is equivalent to screen co-ordinate *x*.

**6.40.2.11 float CEGUI::CoordConverter::screenToWindowY (const Window & *window*, const UDim & *y*) [static]**

Convert a screen relative [UDim](#) co-ordinate value to a window co-ordinate value, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*y* [UDim](#) y co-ordinate value to be converted

**Returns:**

float value describing a window co-ordinate value that is equivalent to screen [UDim](#) co-ordinate *y*.

**6.40.2.12 float CEGUI::CoordConverter::screenToWindowY (const Window & *window*, const float *y*) [static]**

Convert a screen pixel co-ordinate value to a window co-ordinate value, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*y* [UDim](#) y co-ordinate value to be converted.

**Returns:**

float value describing a window co-ordinate value that is equivalent to screen co-ordinate *y*.

**6.40.2.13 Vector2 CEGUI::CoordConverter::screenToWindow (const Window & *window*, const UVector2 & *vec*) [static]**

Convert a screen relative [UVector2](#) point to a window co-ordinate point, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*vec* [UVector2](#) object describing the point to be converted

**Returns:**

[Vector2](#) object describing a window co-ordinate point that is equivalent to screen based [UVector2](#) point *vec*.

**6.40.2.14 Vector2 CEGUI::CoordConverter::screenToWindow (const Window & *window*, const Vector2 & *vec*) [static]**

Convert a screen [Vector2](#) pixel point to a window co-ordinate point, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*vec* [Vector2](#) object describing the point to be converted.

**Returns:**

[Vector2](#) object describing a window co-ordinate point that is equivalent to screen based [Vector2](#) point *vec*.

**6.40.2.15 Rect CEGUI::CoordConverter::screenToWindow (const Window & window, const URect & rect) [static]**

Convert a [URect](#) screen area to a window area, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*rect* [URect](#) object describing the area to be converted

**Returns:**

[Rect](#) object describing a window area that is equivalent to [URect](#) screen area *rect*.

**6.40.2.16 Rect CEGUI::CoordConverter::screenToWindow (const Window & window, const Rect & rect) [static]**

Convert a [Rect](#) screen pixel area to a window area, specified in pixels.

**Parameters:**

*window* [Window](#) object to use as a target for the conversion.

*rect* [Rect](#) object describing the area to be converted.

**Returns:**

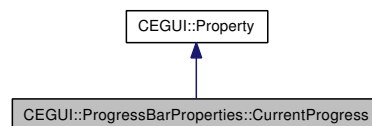
[Rect](#) object describing a window area that is equivalent to [Rect](#) screen area *rect*.



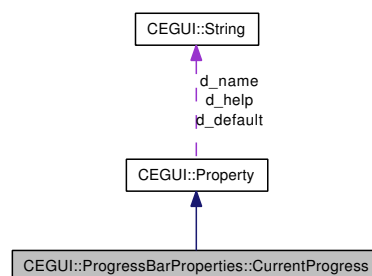
## 6.41 CEGUI::ProgressBarProperties::CurrentProgress Class Reference

[Property](#) to access the current progress of the progress bar.

Inheritance diagram for CEGUI::ProgressBarProperties::CurrentProgress:



Collaboration diagram for CEGUI::ProgressBarProperties::CurrentProgress:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.41.1 Detailed Description

[Property](#) to access the current progress of the progress bar.

##### Usage:

- Name: [CurrentProgress](#)
- Format: "[float]".

##### Where:

- [float] is the current progress of the bar expressed as a value between 0 and 1.

## 6.41.2 Member Function Documentation

### 6.41.2.1 `String CEGUI::ProgressBarProperties::CurrentProgress::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.41.2.2 `void CEGUI::ProgressBarProperties::CurrentProgress::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

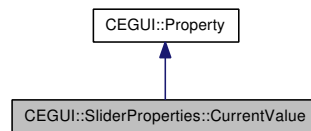
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

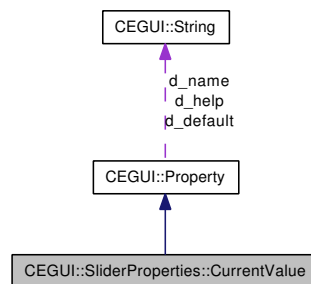
## 6.42 CEGUI::SliderProperties::CurrentValue Class Reference

[Property](#) to access the current value of the slider.

Inheritance diagram for CEGUI::SliderProperties::CurrentValue:



Collaboration diagram for CEGUI::SliderProperties::CurrentValue:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.42.1 Detailed Description

[Property](#) to access the current value of the slider.

**Usage:**

- Name: [CurrentValue](#)
- Format: "[float]".

**Where:**

- [float] represents the current value of the slider.

## 6.42.2 Member Function Documentation

### 6.42.2.1 `String CEGUI::SliderProperties::CurrentValue::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.42.2.2 `void CEGUI::SliderProperties::CurrentValue::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

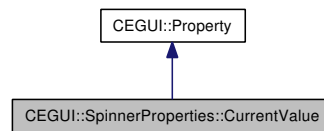
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

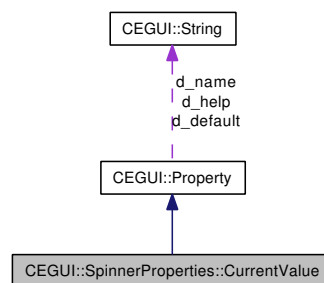
## 6.43 CEGUI::SpinnerProperties::CurrentValue Class Reference

[Property](#) to access the current value of the spinner.

Inheritance diagram for CEGUI::SpinnerProperties::CurrentValue:



Collaboration diagram for CEGUI::SpinnerProperties::CurrentValue:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.43.1 Detailed Description

[Property](#) to access the current value of the spinner.

##### Usage:

- Name: [CurrentValue](#)
- Format: "[float]".

##### Where:

- [float] represents the current value of the [Spinner](#) widget.

## 6.43.2 Member Function Documentation

### 6.43.2.1 `String CEGUI::SpinnerProperties::CurrentValue::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.43.2.2 `void CEGUI::SpinnerProperties::CurrentValue::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

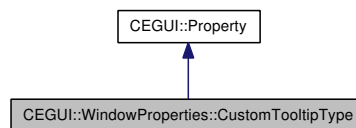
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

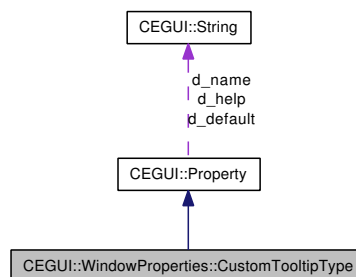
## 6.44 CEGUI::WindowProperties::CustomTooltipType Class Reference

[Property](#) to access the custom tooltip for this [Window](#).

Inheritance diagram for CEGUI::WindowProperties::CustomTooltipType:



Collaboration diagram for CEGUI::WindowProperties::CustomTooltipType:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.44.1 Detailed Description

[Property](#) to access the custom tooltip for this [Window](#).

##### Usage:

- Name: [CustomTooltipType](#)
- Format: "[text]".

##### Where:

- [\[Text\]](#) is the typename of the custom tooltip for the [Window](#).

## 6.44.2 Member Function Documentation

### 6.44.2.1 `String CEGUI::WindowProperties::CustomTooltipType::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.44.2.2 `void CEGUI::WindowProperties::CustomTooltipType::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

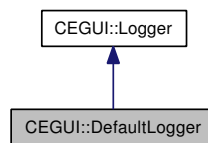
Implements [CEGUI::Property](#).



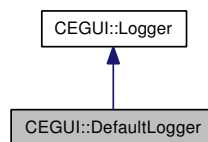
## 6.45 CEGUI::DefaultLogger Class Reference

Default implementation for the [Logger](#) class. If you want to redirect [CEGUI](#) logs to some place other than a text file, implement your own [Logger](#) implementation and create a object of the [Logger](#) type before creating the [CEGUI::System](#) singleton.

Inheritance diagram for CEGUI::DefaultLogger:



Collaboration diagram for CEGUI::DefaultLogger:



### Public Member Functions

- [DefaultLogger](#) (void)  
*Constructor for [DefaultLogger](#) object.*
- virtual [~DefaultLogger](#) (void)  
*Destructor for [DefaultLogger](#) object.*
- virtual void [logEvent](#) (const [String](#) &message, [LoggingLevel](#) level=Standard)  
*Add an event to the log.*
- virtual void [setLogFilename](#) (const [String](#) &filename, bool append=false)  
*Set the name of the log file where all subsequent log entries should be written.*

### Protected Attributes

- std::ofstream [d\\_ostream](#)  
*Stream used to implement the logger.*
- std::vector< std::pair< [String](#), [LoggingLevel](#) > > [d\\_cache](#)  
*Used to cache log entries before log file is created.*
- std::ostringstream [d\\_workstream](#)  
*Used to build log entry strings.*
- bool [d\\_caching](#)

*true while log entries are beign cached (prior to logfile creation)*

### 6.45.1 Detailed Description

Default implementation for the [Logger](#) class. If you want to redirect [CEGUI](#) logs to some place other than a text file, implement your own [Logger](#) implementation and create a object of the [Logger](#) type before creating the [CEGUI::System](#) singleton.

### 6.45.2 Member Function Documentation

#### 6.45.2.1 void CEGUI::DefaultLogger::logEvent (const String & *message*, LoggingLevel *level* = Standard) [virtual]

Add an event to the log.

##### Parameters:

*message* [String](#) object containing the message to be added to the event log.

*level* LoggingLevel for this message. If *level* is greater than the current set logging level, the message is not logged.

##### Returns:

Nothing

Implements [CEGUI::Logger](#).

#### 6.45.2.2 void CEGUI::DefaultLogger::setLogFilename (const String & *filename*, bool *append* = false) [virtual]

Set the name of the log file where all subsequent log entries should be written.

##### Note:

When this is called, and the log file is created, any cached log entries are flushed to the log file.

##### Parameters:

*filename* Name of the file to put log messages.

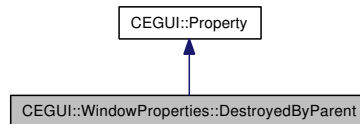
*append*   • true if events should be added to the end of the current file.  
          • false if the current contents of the file should be discarded.

Implements [CEGUI::Logger](#).

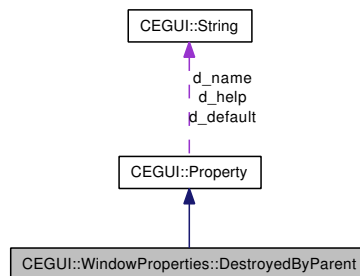
## 6.46 CEGUI::WindowProperties::DestroyedByParent Class Reference

[Property](#) to access window Destroyed by Parent setting.

Inheritance diagram for CEGUI::WindowProperties::DestroyedByParent:



Collaboration diagram for CEGUI::WindowProperties::DestroyedByParent:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.46.1 Detailed Description

[Property](#) to access window Destroyed by Parent setting.

This property offers access to the destroyed by parent setting for the window.

**Usage:**

- Name: [DestroyedByParent](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the [Window](#) should be automatically destroyed when it's parent [Window](#) is destroyed.
- "False" to indicate the [Window](#) should not be destroyed when it's parent [Window](#) is destroyed.

## 6.46.2 Member Function Documentation

### 6.46.2.1 `String CEGUI::WindowProperties::DestroyedByParent::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.46.2.2 `void CEGUI::WindowProperties::DestroyedByParent::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

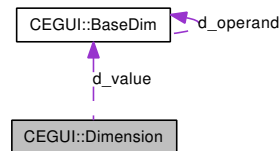
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.47 CEGUI::Dimension Class Reference

Class representing some kind of dimension.

Collaboration diagram for CEGUI::Dimension:



### Public Member Functions

- [Dimension](#) ()  
*Constructor.*
- [~Dimension](#) ()  
*Destructor.*
- [Dimension](#) (const [BaseDim](#) &dim, [DimensionType](#) type)  
*Constructor.*
- [Dimension](#) (const [Dimension](#) &other)  
*Copy constructor.*
- [Dimension](#) & [operator=](#) (const [Dimension](#) &other)  
*Assignment operator.*
- const [BaseDim](#) & [getBaseDimension](#) () const  
*return the [BaseDim](#) object currently used as the value for this [Dimension](#).*
- void [setBaseDimension](#) (const [BaseDim](#) &dim)  
*set the current value for this [Dimension](#).*
- [DimensionType](#) [getDimensionType](#) () const  
*Return a [DimensionType](#) value indicating what this [Dimension](#) represents.*
- void [setDimensionType](#) ([DimensionType](#) type)  
*Sets what this [Dimension](#) represents.*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [Dimension](#) to out\_stream.*

### 6.47.1 Detailed Description

Class representing some kind of dimension.

The key thing to understand about [Dimension](#) is that it contains not just a dimensional value, but also a record of what the dimension value is supposed to represent. (e.g. a co-ordinate on the x axis, or the height of something).

## 6.47.2 Constructor & Destructor Documentation

### 6.47.2.1 CEGUI::Dimension::Dimension (const BaseDim & *dim*, DimensionType *type*)

Constructor.

#### Parameters:

*dim* object based on subclass of [BaseDim](#) which holds the dimensional value.

*type* DimensionType value indicating what dimension this object is to represent.

## 6.47.3 Member Function Documentation

### 6.47.3.1 const BaseDim & CEGUI::Dimension::getBaseDimension () const

return the [BaseDim](#) object currently used as the value for this [Dimension](#).

#### Returns:

const reference to the [BaseDim](#) sub-class object which contains the value for this [Dimension](#).

### 6.47.3.2 void CEGUI::Dimension::setBaseDimension (const BaseDim & *dim*)

set the current value for this [Dimension](#).

#### Parameters:

*dim* object based on a subclass of [BaseDim](#) which holds the dimensional value.

#### Returns:

Nothing.

### 6.47.3.3 DimensionType CEGUI::Dimension::getDimensionType () const

Return a DimensionType value indicating what this [Dimension](#) represents.

#### Returns:

one of the DimensionType enumerated values.

### 6.47.3.4 void CEGUI::Dimension::setDimensionType (DimensionType *type*)

Sets what this [Dimension](#) represents.

**Parameters:**

*type* one of the DimensionType enumerated values.

**Returns:**

Nothing.

**6.47.3.5 void CEGUI::Dimension::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [Dimension](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

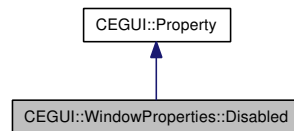
**Returns:**

Nothing.

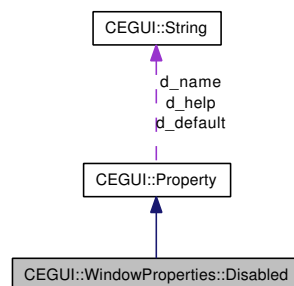
## 6.48 CEGUI::WindowProperties::Disabled Class Reference

[Property](#) to access window [Disabled](#) setting.

Inheritance diagram for CEGUI::WindowProperties::Disabled:



Collaboration diagram for CEGUI::WindowProperties::Disabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- bool [isDefault](#) (const [PropertyReceiver](#) \*receiver) const  
*Returns whether the property is at it's default value.*

### 6.48.1 Detailed Description

[Property](#) to access window [Disabled](#) setting.

This property offers access to the enabled / disabled setting for the window.

**Usage:**

- Name: [Disabled](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the [Window](#) is disabled, and will normally receive no inputs from the user.
- "False" to indicate the [Window](#) is not disabled and will receive inputs from the user as normal.



## 6.48.2 Member Function Documentation

### 6.48.2.1 String CEGUI::WindowProperties::Disabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.48.2.2 void CEGUI::WindowProperties::Disabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

### 6.48.2.3 bool CEGUI::WindowProperties::Disabled::isDefault (const PropertyReceiver \* *receiver*) const [virtual]

Returns whether the property is at it's default value.

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

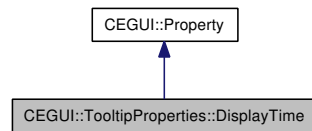
- true if the property has it's default value.
- false if the property has been modified from it's default value.

Reimplemented from [CEGUI::Property](#).

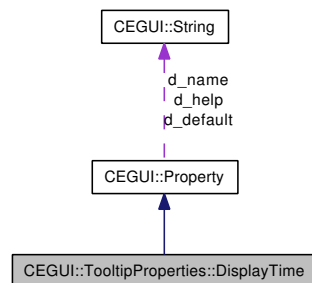
## 6.49 CEGUI::TooltipProperties::DisplayTime Class Reference

[Property](#) to access the time after which the tooltip automatically de-activates itself.

Inheritance diagram for CEGUI::TooltipProperties::DisplayTime:



Collaboration diagram for CEGUI::TooltipProperties::DisplayTime:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.49.1 Detailed Description

[Property](#) to access the time after which the tooltip automatically de-activates itself.

##### Usage:

- Name: [DisplayTime](#)
- Format: "[float]".

##### Where:

- [float] specifies the number of seconds after which the tooltip will deactivate itself if the mouse has remained stationary.

## 6.49.2 Member Function Documentation

### 6.49.2.1 String CEGUI::TooltipProperties::DisplayTime::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.49.2.2 void CEGUI::TooltipProperties::DisplayTime::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

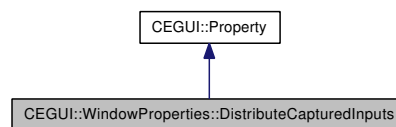
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

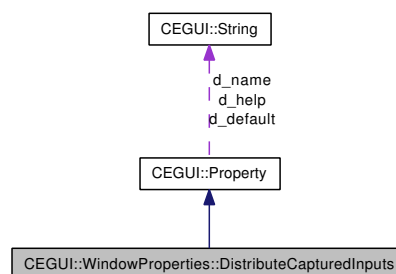
## 6.50 CEGUI::WindowProperties::DistributeCapturedInputs Class Reference

[Property](#) to access whether inputs are passed to child windows when input is captured to this window.

Inheritance diagram for CEGUI::WindowProperties::DistributeCapturedInputs:



Collaboration diagram for CEGUI::WindowProperties::DistributeCapturedInputs:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.50.1 Detailed Description

[Property](#) to access whether inputs are passed to child windows when input is captured to this window.

##### Usage:

- Name: [DistributeCapturedInputs](#)
- Format: "[text]".

##### Where [Text] is:

- "True" to indicate 'captured' inputs should be passed to attached child windows.
- "False" to indicate 'captured' inputs should be passed to this window only.

## 6.50.2 Member Function Documentation

### 6.50.2.1 String CEGUI::WindowProperties::DistributeCapturedInputs::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.50.2.2 void CEGUI::WindowProperties::DistributeCapturedInputs::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

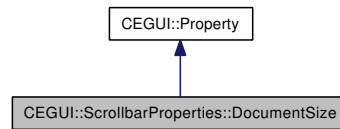
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

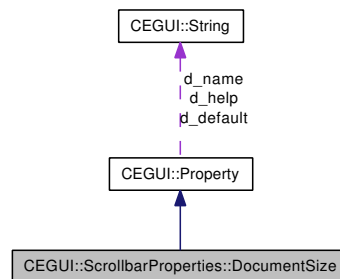
## 6.51 CEGUI::ScrollbarProperties::DocumentSize Class Reference

[Property](#) to access the document size for the [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollbarProperties::DocumentSize:



Collaboration diagram for CEGUI::ScrollbarProperties::DocumentSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.51.1 Detailed Description

[Property](#) to access the document size for the [Scrollbar](#).

##### Usage:

- Name: [DocumentSize](#)
- Format: "[float]".

##### Where:

- [float] specifies the size of the document being scrolled (as defined by the client code).

## 6.51.2 Member Function Documentation

### 6.51.2.1 String CEGUI::ScrollbarProperties::DocumentSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.51.2.2 void CEGUI::ScrollbarProperties::DocumentSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

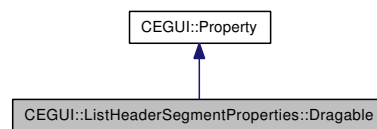
[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

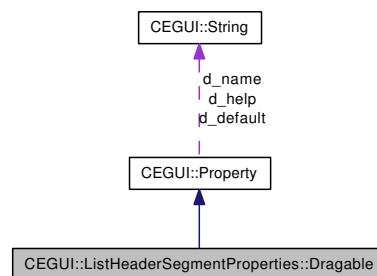
## 6.52 CEGUI::ListHeaderSegmentProperties::Dragable Class Reference

[Property](#) to access the drag-able setting of the header segment.

Inheritance diagram for CEGUI::ListHeaderSegmentProperties::Dragable:



Collaboration diagram for CEGUI::ListHeaderSegmentProperties::Dragable:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.52.1 Detailed Description

[Property](#) to access the drag-able setting of the header segment.

Usage:

- Name: [Dragable](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the segment can be dragged by the user.
- "False" to indicate the segment can not be dragged by the user.



## 6.52.2 Member Function Documentation

### 6.52.2.1 String CEGUI::ListHeaderSegmentProperties::Dragable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.52.2.2 void CEGUI::ListHeaderSegmentProperties::Dragable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

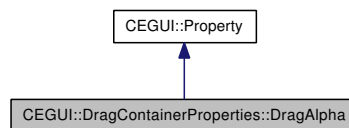
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

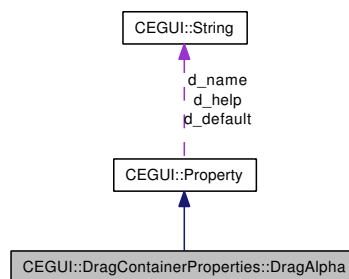
## 6.53 CEGUI::DragContainerProperties::DragAlpha Class Reference

[Property](#) to access the dragging alpha value.

Inheritance diagram for CEGUI::DragContainerProperties::DragAlpha:



Collaboration diagram for CEGUI::DragContainerProperties::DragAlpha:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.53.1 Detailed Description

[Property](#) to access the dragging alpha value.

##### Usage:

- Name: [DragAlpha](#)
- Format: "[float]".

##### Where:

- [float] represents the alpha value to set when dragging.

## 6.53.2 Member Function Documentation

### 6.53.2.1 String CEGUI::DragContainerProperties::DragAlpha::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.53.2.2 void CEGUI::DragContainerProperties::DragAlpha::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

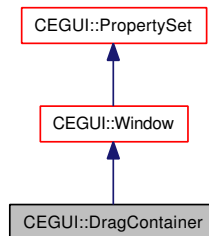
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

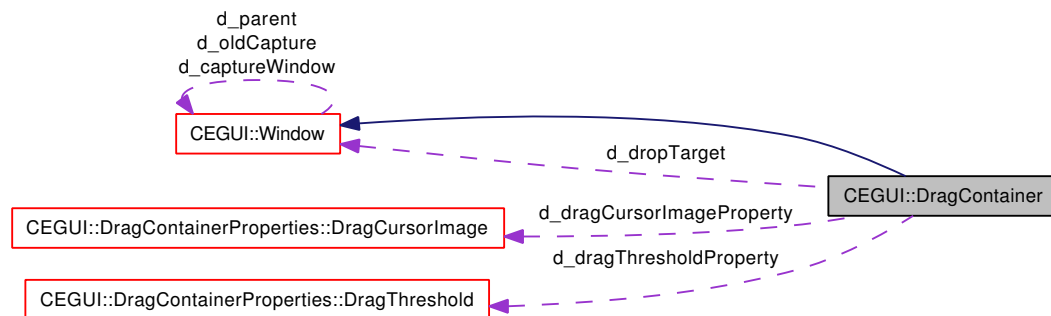
## 6.54 CEGUI::DragContainer Class Reference

Generic drag & drop enabled window class.

Inheritance diagram for CEGUI::DragContainer:



Collaboration diagram for CEGUI::DragContainer:



### Public Member Functions

- **DragContainer** (const [String](#) &type, const [String](#) &name)  
*Constructor for [DragContainer](#) objects.*
- virtual **~DragContainer** (void)  
*Destructor for [DragContainer](#) objects.*
- bool **isDraggingEnabled** (void) const  
*Return whether dragging is currently enabled for this [DragContainer](#).*
- void **setDraggingEnabled** (bool setting)  
*Set whether dragging is currently enabled for this [DragContainer](#).*
- bool **isBeingDragged** (void) const  
*Return whether the [DragContainer](#) is currently being dragged.*
- float **getPixelDragThreshold** (void) const  
*Return the current drag threshold in pixels.*
- void **setPixelDragThreshold** (float pixels)

*Set the current drag threshold in pixels.*

- float [getDragAlpha](#) (void) const  
*Return the alpha value that will be set on the [DragContainer](#) while a drag operation is in progress.*
- void [setDragAlpha](#) (float alpha)  
*Set the alpha value to be set on the [DragContainer](#) when a drag operation is in progress.*
- const [Image](#) \* [getDragCursorImage](#) (void) const  
*Return the [Image](#) currently set to be used for the mouse cursor when a drag operation is in progress.*
- void [setDragCursorImage](#) (const [Image](#) \*image)  
*Set the [Image](#) to be used for the mouse cursor when a drag operation is in progress.*
- void [setDragCursorImage](#) ([MouseCursorImage](#) image)  
*Set the [Image](#) to be used for the mouse cursor when a drag operation is in progress.*
- void [setDragCursorImage](#) (const [String](#) &imageset, const [String](#) &image)  
*Set the [Image](#) to be used for the mouse cursor when a drag operation is in progress.*
- [Window](#) \* [getCurrentDropTarget](#) (void) const  
*Return the [Window](#) object that is the current drop target for the [DragContainer](#).*

## Static Public Attributes

- static const [String](#) [WidgetTypeName](#)  
*Type name for [DragContainer](#).*
- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [EventDragStarted](#)  
*Name of the event fired when the user begins dragging the thumb.*
- static const [String](#) [EventDragEnded](#)  
*Name of the event fired when the user releases the thumb.*
- static const [String](#) [EventDragPositionChanged](#)  
*Event fired when the drag position has changed.*
- static const [String](#) [EventDragEnabledChanged](#)  
*Event fired when dragging is enabled or disabled.*
- static const [String](#) [EventDragAlphaChanged](#)  
*Event fired when the alpha value used when dragging is changed.*
- static const [String](#) [EventDragMouseCursorChanged](#)  
*Event fired when the mouse cursor used when dragging is changed.*

- static const [String EventDragThresholdChanged](#)  
*Event fired when the drag pixel threshold is changed.*
- static const [String EventDragDropTargetChanged](#)  
*Event fired when the drop target changes.*

## Protected Member Functions

- bool [isDraggingThresholdExceeded](#) (const [Point](#) &local\_mouse)  
*Return whether the required minimum movement threshold before initiating dragging has been exceeded.*
- void [initialiseDragging](#) (void)  
*Initialise the required states to put the window into dragging mode.*
- void [doDragging](#) (const [Point](#) &local\_mouse)  
*Update state for window dragging.*
- void [updateActiveMouseCursor](#) (void) const  
*Method to update mouse cursor image.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [onMouseButtonDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseButtonUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void [onAlphaChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's alpha blend value is changed.*
- virtual void [onClippingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's setting for being clipped by it's parent is changed.*
- virtual void [onMoved](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's position changes.*
- virtual void [onDragStarted](#) ([WindowEventArgs](#) &e)  
*Method called when dragging commences.*
- virtual void [onDragEnded](#) ([WindowEventArgs](#) &e)

*Method called when dragging ends.*

- virtual void [onDragPositionChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the dragged object position is changed.*
- virtual void [onDragEnabledChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the dragging state is enabled or disabled.*
- virtual void [onDragAlphaChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the alpha value to use when dragging is changed.*
- virtual void [onDragMouseCursorChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the mouse cursor to use when dragging is changed.*
- virtual void [onDragThresholdChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the movement threshold required to trigger dragging is changed.*
- virtual void [onDragDropTargetChanged](#) ([DragDropEventArgs](#) &e)  
*Method called when the current drop target of this [DragContainer](#) changes.*

## Protected Attributes

- bool [d\\_draggingEnabled](#)  
*True when dragging is enabled.*
- bool [d\\_leftMouseDown](#)  
*True when left mouse button is down.*
- bool [d\\_dragging](#)  
*true when being dragged.*
- [UVector2](#) [d\\_dragPoint](#)  
*point we are being dragged at.*
- [UVector2](#) [d\\_startPosition](#)  
*position prior to dragging.*
- float [d\\_dragThreshold](#)  
*Pixels mouse must move before dragging commences.*
- float [d\\_dragAlpha](#)  
*Alpha value to set when dragging.*
- float [d\\_storedAlpha](#)  
*Alpha value to re-set when dragging ends.*
- bool [d\\_storedClipState](#)  
*Parent clip state to re-set.*

- [Window](#) \* [d\\_dropTarget](#)  
*Target window for possible drop operation.*
- const [Image](#) \* [d\\_dragCursorImage](#)  
*Image to use for mouse cursor when dragging.*
- bool [d\\_dropflag](#)  
*True when we're being dropped.*

### 6.54.1 Detailed Description

Generic drag & drop enabled window class.

### 6.54.2 Member Function Documentation

#### 6.54.2.1 bool CEGUI::DragContainer::isDraggingEnabled (void) const

Return whether dragging is currently enabled for this [DragContainer](#).

**Returns:**

- true if dragging is enabled and the [DragContainer](#) may be dragged.
- false if dragging is disabled and the [DragContainer](#) may not be dragged.

#### 6.54.2.2 void CEGUI::DragContainer::setDraggingEnabled (bool *setting*)

Set whether dragging is currently enabled for this [DragContainer](#).

**Parameters:**

- setting*
- true to enable dragging so that the [DragContainer](#) may be dragged.
  - false to disabled dragging so that the [DragContainer](#) may not be dragged.

**Returns:**

Nothing.

#### 6.54.2.3 bool CEGUI::DragContainer::isBeingDragged (void) const

Return whether the [DragContainer](#) is currently being dragged.

**Returns:**

- true if the [DragContainer](#) is being dragged.
- false if te [DragContainer](#) is not being dragged.



**6.54.2.4 float CEGUI::DragContainer::getPixelDragThreshold (void) const**

Return the current drag threshold in pixels.

The drag threshold is the number of pixels that the mouse must be moved with the left button held down in order to commence a drag operation.

**Returns:**

float value indicating the current drag threshold value.

**6.54.2.5 void CEGUI::DragContainer::setPixelDragThreshold (float *pixels*)**

Set the current drag threshold in pixels.

The drag threshold is the number of pixels that the mouse must be moved with the left button held down in order to commence a drag operation.

**Parameters:**

*pixels* float value indicating the new drag threshold value.

**Returns:**

Nothing.

**6.54.2.6 float CEGUI::DragContainer::getDragAlpha (void) const**

Return the alpha value that will be set on the [DragContainer](#) while a drag operation is in progress.

**Returns:**

Current alpha value to use whilst dragging.

**6.54.2.7 void CEGUI::DragContainer::setDragAlpha (float *alpha*)**

Set the alpha value to be set on the [DragContainer](#) when a drag operation is in progress.

This method can be used while a drag is in progress to update the alpha. Note that the normal setAlpha method does not affect alpha while a drag is in progress, but once the drag operation has ended, any value set via setAlpha will be restored.

**Parameters:**

*alpha* Alpha value to use whilst dragging.

**Returns:**

Nothing.

**6.54.2.8    `const Image * CEGUI::DragContainer::getDragCursorImage (void) const`**

Return the [Image](#) currently set to be used for the mouse cursor when a drag operation is in progress.

**Returns:**

[Image](#) object currently set to be used as the mouse cursor when dragging.

**6.54.2.9    `void CEGUI::DragContainer::setDragCursorImage (const Image * image)`**

Set the [Image](#) to be used for the mouse cursor when a drag operation is in progress.

This method may be used during a drag operation to update the current mouse cursor image.

**Parameters:**

*image* [Image](#) object to be used as the mouse cursor while dragging.

**Returns:**

Nothing.

**6.54.2.10    `void CEGUI::DragContainer::setDragCursorImage (MouseButton image)`**

Set the [Image](#) to be used for the mouse cursor when a drag operation is in progress.

This method may be used during a drag operation to update the current mouse cursor image.

**Parameters:**

*image* One of the MouseButton enumerated values.

**Returns:**

Nothing.

**6.54.2.11    `void CEGUI::DragContainer::setDragCursorImage (const String & imageset, const String & image)`**

Set the [Image](#) to be used for the mouse cursor when a drag operation is in progress.

This method may be used during a drag operation to update the current mouse cursor image.

**Parameters:**

*imageset* [String](#) holding the name of the [Imageset](#) that contains the [Image](#) to be used.

*image* [Image](#) defined for the [Imageset](#) *imageset* to be used as the mouse cursor when dragging.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if either *imageset* or *image* are unknown.

**6.54.2.12 Window \* CEGUI::DragContainer::getCurrentDropTarget (void) const**

Return the [Window](#) object that is the current drop target for the [DragContainer](#).

The drop target for a [DragContainer](#) is basically the [Window](#) that the [DragContainer](#) is within while being dragged. The drop target may be 0 to indicate no target.

**Returns:**

Pointer to a [Window](#) object that contains the [DragContainer](#) whilst being dragged, or 0 to indicate no current target.

**6.54.2.13 bool CEGUI::DragContainer::isDraggingThresholdExceeded (const Point & local\_mouse) [protected]**

Return whether the required minimum movement threshold before initiating dragging has been exceeded.

**Parameters:**

*local\_mouse* Mouse position as a pixel offset from the top-left corner of this window.

**Returns:**

- true if the threshold has been exceeded and dragging should be initiated.
- false if the threshold has not been exceeded.

**6.54.2.14 void CEGUI::DragContainer::initialiseDragging (void) [protected]**

Initialise the required states to put the window into dragging mode.

**Returns:**

Nothing.

**6.54.2.15 void CEGUI::DragContainer::doDragging (const Point & local\_mouse) [protected]**

Update state for window dragging.

**Parameters:**

*local\_mouse* Mouse position as a pixel offset from the top-left corner of this window.

**Returns:**

Nothing.

**6.54.2.16** `virtual bool CEGUI::DragContainer::testClassName_impl (const String & class_name)`  
`const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.54.2.17** `void CEGUI::DragContainer::onMouseButtonDown (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.54.2.18** `void CEGUI::DragContainer::onMouseButtonUp (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.54.2.19** `void CEGUI::DragContainer::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.54.2.20 void CEGUI::DragContainer::onCaptureLost (WindowEventArgs & e)**  
[protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.54.2.21 void CEGUI::DragContainer::onAlphaChanged (WindowEventArgs & e)**  
[protected, virtual]

Handler called when the window's alpha blend value is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.54.2.22 void CEGUI::DragContainer::onClippingChanged (WindowEventArgs & e)**  
[protected, virtual]

Handler called when the window's setting for being clipped by it's parent is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.54.2.23 void CEGUI::DragContainer::onMoved (WindowEventArgs & e)** [protected, virtual]

Handler called when the window's position changes.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.54.2.24 void CEGUI::DragContainer::onDragStarted (WindowEventArgs & e)**  
[protected, virtual]

Method called when dragging commences.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.54.2.25 void CEGUI::DragContainer::onDragEnded (WindowEventArgs & *e*)** [protected, virtual]

Method called when dragging ends.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.54.2.26 void CEGUI::DragContainer::onDragPositionChanged (WindowEventArgs & *e*)** [protected, virtual]

Method called when the dragged object position is changed.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.54.2.27 void CEGUI::DragContainer::onDragEnabledChanged (WindowEventArgs & *e*)** [protected, virtual]

Method called when the dragging state is enabled or disabled.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.54.2.28 void CEGUI::DragContainer::onDragAlphaChanged (WindowEventArgs & *e*)**  
[protected, virtual]

Method called when the alpha value to use when dragging is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.54.2.29 void CEGUI::DragContainer::onDragMouseCursorChanged (WindowEventArgs & *e*)**  
[protected, virtual]

Method called when the mouse cursor to use when dragging is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.54.2.30 void CEGUI::DragContainer::onDragThresholdChanged (WindowEventArgs & *e*)**  
[protected, virtual]

Method called when the movement threshold required to trigger dragging is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.54.2.31 void CEGUI::DragContainer::onDragDropTargetChanged (DragDropEventArgs & *e*)**  
[protected, virtual]

Method called when the current drop target of this [DragContainer](#) changes.

**Note:**

This event fires just prior to the target field being changed. The default implementation changes the drop target, you can examine the old and new targets before calling the default implementation to make the actual change (and fire appropriate events for the [Window](#) objects involved).

**Parameters:**

*e* [DragDropEventArgs](#) object initialised as follows:

- dragDropItem is initialised to the [DragContainer](#) triggering the event (typically 'this').
- window is initialised to point to the [Window](#) which will be the new drop target.

**Returns:**

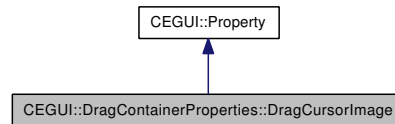
Nothing.



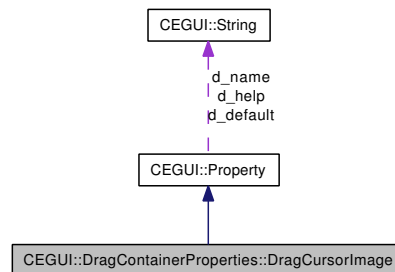
## 6.55 CEGUI::DragContainerProperties::DragCursorImage Class Reference

[Property](#) to access the dragging mouse cursor setting.

Inheritance diagram for CEGUI::DragContainerProperties::DragCursorImage:



Collaboration diagram for CEGUI::DragContainerProperties::DragCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.55.1 Detailed Description

[Property](#) to access the dragging mouse cursor setting.

This property offers access to the mouse cursor image used when dragging the [DragContainer](#).

#### Usage:

- Name: [DragCursorImage](#)
- Format: "set:[text] image:[text]".

#### Where:

- set:[text] is the name of the [Imageset](#) containing the image. The [Imageset](#) name should not contain spaces. The [Imageset](#) specified must already be loaded.
- image:[text] is the name of the [Image](#) on the specified [Imageset](#). The [Image](#) name should not contain spaces.

## 6.55.2 Member Function Documentation

### 6.55.2.1 String CEGUI::DragContainerProperties::DragCursorImage::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.55.2.2 void CEGUI::DragContainerProperties::DragCursorImage::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

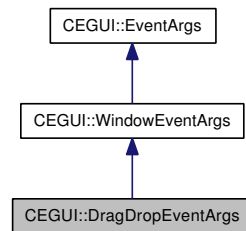
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

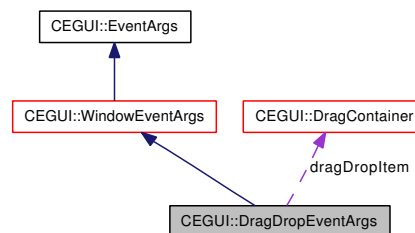
## 6.56 CEGUI::DragDropEventArgs Class Reference

[EventArgs](#) based class used for certain drag/drop notifications.

Inheritance diagram for CEGUI::DragDropEventArgs:



Collaboration diagram for CEGUI::DragDropEventArgs:



### Public Member Functions

- [DragDropEventArgs](#) ([Window](#) \*wnd)

### Public Attributes

- [DragContainer](#) \* [dragDropItem](#)

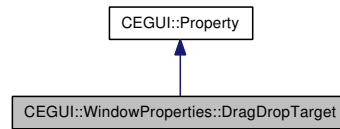
#### 6.56.1 Detailed Description

[EventArgs](#) based class used for certain drag/drop notifications.

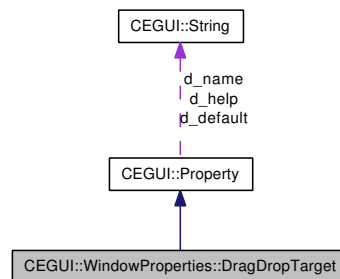
## 6.57 CEGUI::WindowProperties::DragDropTarget Class Reference

[Property](#) to get/set whether the [Window](#) will receive drag and drop related notifications.

Inheritance diagram for CEGUI::WindowProperties::DragDropTarget:



Collaboration diagram for CEGUI::WindowProperties::DragDropTarget:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.57.1 Detailed Description

[Property](#) to get/set whether the [Window](#) will receive drag and drop related notifications.

**Usage:**

- Name: [DragDropTarget](#)
- Format: "[text]".

**Where [Text] is:**

- "True" if [Window](#) is will receive drag & drop notifications.
- "False" if [Window](#) is will not receive drag & drop notifications.

## 6.57.2 Member Function Documentation

### 6.57.2.1 String CEGUI::WindowProperties::DragDropTarget::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.57.2.2 void CEGUI::WindowProperties::DragDropTarget::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

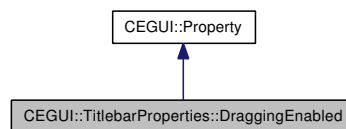
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

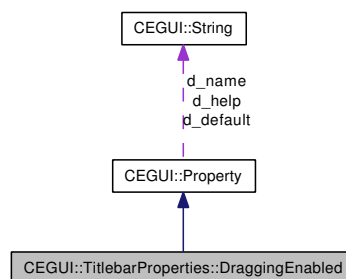
## 6.58 CEGUI::TitlebarProperties::DraggingEnabled Class Reference

[Property](#) to access the state of the dragging enabled setting for the [Titlebar](#).

Inheritance diagram for CEGUI::TitlebarProperties::DraggingEnabled:



Collaboration diagram for CEGUI::TitlebarProperties::DraggingEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.58.1 Detailed Description

[Property](#) to access the state of the dragging enabled setting for the [Titlebar](#).

Usage:

- Name: [DraggingEnabled](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate that drag moving is enabled.
- "False" to indicate that drag moving is disabled.

## 6.58.2 Member Function Documentation

### 6.58.2.1 String CEGUI::TitlebarProperties::DraggingEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.58.2.2 void CEGUI::TitlebarProperties::DraggingEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

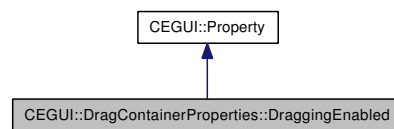
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

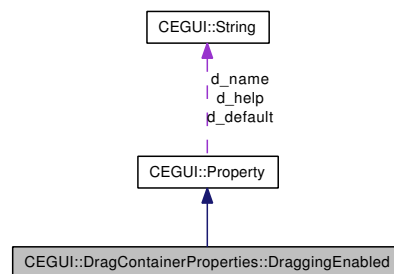
## 6.59 CEGUI::DragContainerProperties::DraggingEnabled Class Reference

[Property](#) to access the state of the dragging enabled setting.

Inheritance diagram for CEGUI::DragContainerProperties::DraggingEnabled:



Collaboration diagram for CEGUI::DragContainerProperties::DraggingEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.59.1 Detailed Description

[Property](#) to access the state of the dragging enabled setting.

##### Usage:

- Name: [DraggingEnabled](#)
- Format: "[text]".

##### Where [Text] is:

- "True" to indicate that dragging is enabled.
- "False" to indicate that dragging is disabled.



## 6.59.2 Member Function Documentation

### 6.59.2.1 String CEGUI::DragContainerProperties::DraggingEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.59.2.2 void CEGUI::DragContainerProperties::DraggingEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

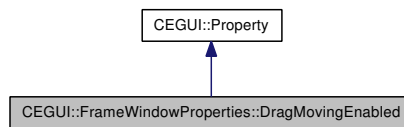
[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

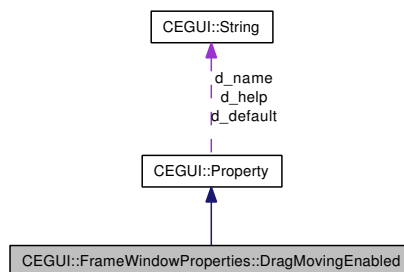
## 6.60 CEGUI::FrameWindowProperties::DragMovingEnabled Class Reference

[Property](#) to access the setting for whether the user may drag the window around by its title bar.

Inheritance diagram for CEGUI::FrameWindowProperties::DragMovingEnabled:



Collaboration diagram for CEGUI::FrameWindowProperties::DragMovingEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.60.1 Detailed Description

[Property](#) to access the setting for whether the user may drag the window around by its title bar.

##### Usage:

- Name: [DragMovingEnabled](#)
- Format: "[text]".

##### Where [Text] is:

- "True" to indicate the window may be repositioned by the user via dragging.
- "False" to indicate the window may not be repositioned by the user.

## 6.60.2 Member Function Documentation

### 6.60.2.1 String CEGUI::FrameWindowProperties::DragMovingEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.60.2.2 void CEGUI::FrameWindowProperties::DragMovingEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

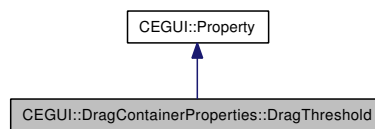
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

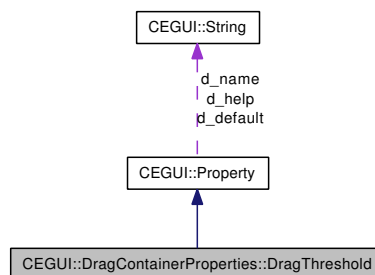
## 6.61 CEGUI::DragContainerProperties::DragThreshold Class Reference

[Property](#) to access the dragging threshold value.

Inheritance diagram for CEGUI::DragContainerProperties::DragThreshold:



Collaboration diagram for CEGUI::DragContainerProperties::DragThreshold:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.61.1 Detailed Description

[Property](#) to access the dragging threshold value.

#### Usage:

- Name: [DragThreshold](#)
- Format: "[float]".

#### Where:

- [float] represents the movement threshold (in pixels) which must be exceeded to commence dragging.

## 6.61.2 Member Function Documentation

### 6.61.2.1 String CEGUI::DragContainerProperties::DragThreshold::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.61.2.2 void CEGUI::DragContainerProperties::DragThreshold::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

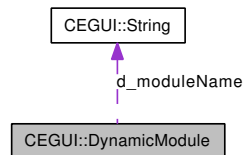
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.62 CEGUI::DynamicModule Class Reference

Class that wraps and gives access to a dynamically linked module (.dll, .so, etc...).

Collaboration diagram for CEGUI::DynamicModule:



### Public Member Functions

- **DynamicModule** (const **String** &name)  
Construct the *DynamicModule* object by loading the dynamic loadable module specified.
- **~DynamicModule** ()  
Destroys the *DynamicModule* object and unloads the associated loadable module.
- const **String** & **getModuleName** () const  
Return a *String* containing the name of the dynamic module.
- void \* **getSymbolAddress** (const **String** &symbol) const  
Return the address of the specified symbol.

### 6.62.1 Detailed Description

Class that wraps and gives access to a dynamically linked module (.dll, .so, etc...).

### 6.62.2 Constructor & Destructor Documentation

#### 6.62.2.1 CEGUI::DynamicModule::DynamicModule (const **String** & name)

Construct the *DynamicModule* object by loading the dynamic loadable module specified.

##### Parameters:

*name* *String* object holding the name of a loadable module.

##### Returns:

Nothing

### 6.62.2.2 CEGUI::DynamicModule::~~DynamicModule ()

Destroys the [DynamicModule](#) object and unloads the associated loadable module.

**Returns:**

Nothing

## 6.62.3 Member Function Documentation

### 6.62.3.1 void \* CEGUI::DynamicModule::getSymbolAddress (const String & *symbol*) const

Return the address of the specified symbol.

**Parameters:**

*symbol* [String](#) holding the symbol to look up in the module.

**Returns:**

Pointer to the requested symbol.

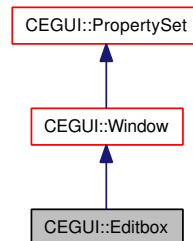
**Exceptions:**

[InvalidRequestException](#) thrown if the symbol does not exist.

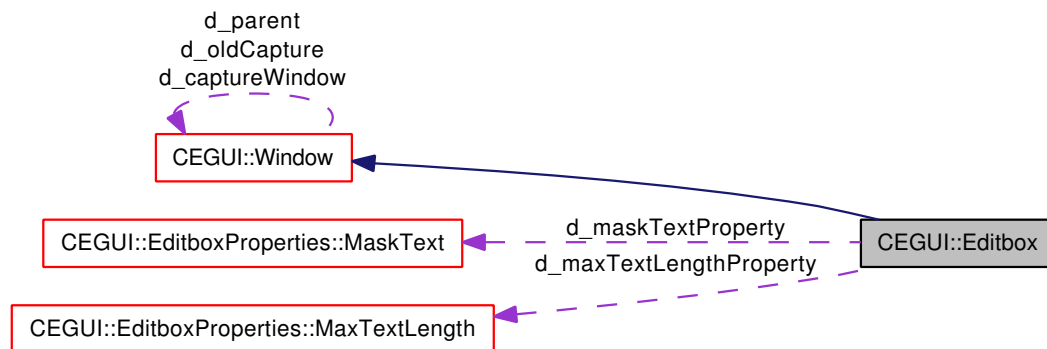
## 6.63 CEGUI::Editbox Class Reference

Base class for an [Editbox](#) widget.

Inheritance diagram for CEGUI::Editbox:



Collaboration diagram for CEGUI::Editbox:



### Public Member Functions

- bool [hasInputFocus](#) (void) const  
*return true if the [Editbox](#) has input focus.*
- bool [isReadOnly](#) (void) const  
*return true if the [Editbox](#) is read-only.*
- bool [isTextMasked](#) (void) const  
*return true if the text for the [Editbox](#) will be rendered masked.*
- bool [isTextValid](#) (void) const  
*return true if the [Editbox](#) text is valid given the currently set validation string.*
- const [String](#) & [getValidationString](#) (void) const  
*return the currently set validation string*
- size\_t [getCaratIndex](#) (void) const  
*return the current position of the carat.*



- `size_t` [getSelectionStartIndex](#) (void) const  
*return the current selection start point.*
- `size_t` [getSelectionEndIndex](#) (void) const  
*return the current selection end point.*
- `size_t` [getSelectionLength](#) (void) const  
*return the length of the current selection (in code points / characters).*
- `utf32` [getMaskCodePoint](#) (void) const  
*return the utf32 code point used when rendering masked text.*
- `size_t` [getMaxTextLength](#) (void) const  
*return the maximum text length set for this [Editbox](#).*
- void [setReadOnly](#) (bool setting)  
*Specify whether the [Editbox](#) is read-only.*
- void [setTextMasked](#) (bool setting)  
*Specify whether the text for the [Editbox](#) will be rendered masked.*
- void [setValidationString](#) (const [String](#) &validation\_string)  
*Set the text validation string.*
- void [setCaratIndex](#) (size\_t carat\_pos)  
*Set the current position of the carat.*
- void [setSelection](#) (size\_t start\_pos, size\_t end\_pos)  
*Define the current selection for the [Editbox](#).*
- void [setMaskCodePoint](#) (utf32 code\_point)  
*set the utf32 code point used when rendering masked text.*
- void [setMaxTextLength](#) (size\_t max\_len)  
*set the maximum text length for this [Editbox](#).*
- [Editbox](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Editbox](#) class.*
- virtual [~Editbox](#) (void)  
*Destructor for [Editbox](#) class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)

*Window* factory name.

- static const [String EventReadOnlyModeChanged](#)  
*The read-only mode for the edit box has been changed.*
- static const [String EventMaskedRenderingModeChanged](#)  
*The masked rendering mode (password mode) has been changed.*
- static const [String EventMaskCodePointChanged](#)  
*The code point (character) to use for masked text has been changed.*
- static const [String EventValidationStringChanged](#)  
*The validation string has been changed.*
- static const [String EventMaximumTextLengthChanged](#)  
*The maximum allowable string length has been changed.*
- static const [String EventTextInvalidated](#)  
*Some operation has made the current text invalid with regards to the validation string.*
- static const [String EventInvalidEntryAttempted](#)  
*The user attempted to modify the text in a way that would have made it invalid.*
- static const [String EventCaratMoved](#)  
*The text carat (insert point) has changed.*
- static const [String EventTextSelectionChanged](#)  
*The current text selection has changed.*
- static const [String EventEditboxFull](#)  
*The number of characters in the edit box has reached the current maximum.*
- static const [String EventTextAccepted](#)  
*The user has accepted the current text by pressing Return, Enter, or Tab.*

## Protected Member Functions

- size\_t [getTextIndexFromPosition](#) (const [Point](#) &pt) const  
*Return the text code point index that is rendered closest to screen position pt.*
- void [clearSelection](#) (void)  
*Return the text code point index that is rendered closest to screen position pt.*
- void [eraseSelectedText](#) (bool modify\_text=true)  
*Erase the currently selected text.*
- bool [isStringValid](#) (const [String](#) &str) const  
*return true if the given string matches the validation regular expression.*

- void [handleBackspace](#) (void)  
*Processing for backspace key.*
- void [handleDelete](#) (void)  
*Processing for Delete key.*
- void [handleCharLeft](#) (uint sysKeys)  
*Processing to move carat one character left.*
- void [handleWordLeft](#) (uint sysKeys)  
*Processing to move carat one word left.*
- void [handleCharRight](#) (uint sysKeys)  
*Processing to move carat one character right.*
- void [handleWordRight](#) (uint sysKeys)  
*Processing to move carat one word right.*
- void [handleHome](#) (uint sysKeys)  
*Processing to move carat to the start of the text.*
- void [handleEnd](#) (uint sysKeys)  
*Processing to move carat to the end of the text.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- virtual void [onReadOnlyChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) fired internally when the read only state of the [Editbox](#) has been changed.*
- virtual void [onMaskedRenderingModeChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) fired internally when the masked rendering mode (password mode) has been changed.*
- virtual void [onMaskCodePointChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) fired internally when the code point to use for masked rendering has been changed.*
- virtual void [onValidationStringChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) fired internally when the validation string is changed.*
- virtual void [onMaximumTextLengthChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) fired internally when the maximum text length for the edit box is changed.*
- virtual void [onTextInvalidatedEvent](#) ([WindowEventArgs](#) &e)  
*[Event](#) fired internally when something has caused the current text to now fail validation.*
- virtual void [onInvalidEntryAttempted](#) ([WindowEventArgs](#) &e)

*Event fired internally when the user attempted to make a change to the edit box that would have caused it to fail validation.*

- virtual void `onCaratMoved` (`WindowEventArgs` &`e`)  
*Event fired internally when the carat (insert point) position changes.*
- virtual void `onTextSelectionChanged` (`WindowEventArgs` &`e`)  
*Event fired internally when the current text selection changes.*
- virtual void `onEditboxFullEvent` (`WindowEventArgs` &`e`)  
*Event fired internally when the edit box text has reached the set maximum length.*
- virtual void `onTextAcceptedEvent` (`WindowEventArgs` &`e`)  
*Event fired internally when the user accepts the edit box text by pressing Return, Enter, or Tab.*
- virtual void `onMouseButtonDown` (`MouseEventArgs` &`e`)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void `onMouseButtonUp` (`MouseEventArgs` &`e`)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void `onMouseDoubleClicked` (`MouseEventArgs` &`e`)  
*Handler called when a mouse button has been double-clicked within this window's area.*
- virtual void `onMouseTripleClicked` (`MouseEventArgs` &`e`)  
*Handler called when a mouse button has been triple-clicked within this window's area.*
- virtual void `onMouseMove` (`MouseEventArgs` &`e`)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void `onCaptureLost` (`WindowEventArgs` &`e`)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void `onCharacter` (`KeyEventArgs` &`e`)  
*Handler called when a character-key has been pressed while this window has input focus.*
- virtual void `onKeyDown` (`KeyEventArgs` &`e`)  
*Handler called when a key as been depressed while this window has input focus.*
- virtual void `onTextChanged` (`WindowEventArgs` &`e`)  
*Handler called when the window's text is changed.*

## Protected Attributes

- bool `d_readOnly`  
*True if the editbox is in read-only mode.*
- bool `d_maskText`  
*True if the editbox text should be rendered masked.*

- utf32 [d\\_maskCodePoint](#)  
*Code point to use when rendering masked text.*
- size\_t [d\\_maxTextLen](#)  
*Maximum number of characters for this [Editbox](#).*
- size\_t [d\\_caratPos](#)  
*Position of the carat / insert-point.*
- size\_t [d\\_selectionStart](#)  
*Start of selection area.*
- size\_t [d\\_selectionEnd](#)  
*End of selection area.*
- String [d\\_validationString](#)  
*Copy of validation reg-ex string.*
- RegexValidator \* [d\\_validator](#)  
*RegEx [String](#) used for validation of text.*
- bool [d\\_dragging](#)  
*true when a selection is being dragged.*
- size\_t [d\\_dragAnchorIdx](#)  
*Selection index for drag selection anchor point.*

### 6.63.1 Detailed Description

Base class for an [Editbox](#) widget.

### 6.63.2 Member Function Documentation

#### 6.63.2.1 bool CEGUI::Editbox::hasInputFocus (void) const

return true if the [Editbox](#) has input focus.

**Returns:**

true if the [Editbox](#) has keyboard input focus, false if the [Editbox](#) does not have keyboard input focus.

#### 6.63.2.2 bool CEGUI::Editbox::isReadOnly (void) const [inline]

return true if the [Editbox](#) is read-only.

**Returns:**

true if the [Editbox](#) is read only and can't be edited by the user, false if the [Editbox](#) is not read only and may be edited by the user.

**6.63.2.3 bool CEGUI::Editbox::isTextMasked (void) const** [inline]

return true if the text for the [Editbox](#) will be rendered masked.

**Returns:**

true if the [Editbox](#) text will be rendered masked using the currently set mask code point, false if the [Editbox](#) text will be rendered as plain text.

**6.63.2.4 bool CEGUI::Editbox::isTextValid (void) const**

return true if the [Editbox](#) text is valid given the currently set validation string.

**Note:**

It is possible to programmatically set 'invalid' text for the [Editbox](#) by calling `setText`. This has certain implications since if invalid text is set, whatever the user types into the box will be rejected when the input is validated.

Validation is performed by means of a regular expression. If the text matches the regex, the text is said to have passed validation. If the text does not match with the regex then the text fails validation.

**Returns:**

true if the current [Editbox](#) text passes validation, false if the text does not pass validation.

**6.63.2.5 const String& CEGUI::Editbox::getValidationString (void) const** [inline]

return the currently set validation string

**Note:**

Validation is performed by means of a regular expression. If the text matches the regex, the text is said to have passed validation. If the text does not match with the regex then the text fails validation.

**Returns:**

[String](#) object containing the current validation regex data

**6.63.2.6 size\_t CEGUI::Editbox::getCaratIndex (void) const** [inline]

return the current position of the carat.

**Returns:**

Index of the insert carat relative to the start of the text.

**6.63.2.7 size\_t CEGUI::Editbox::getSelectionStartIndex (void) const**

return the current selection start point.

**Returns:**

Index of the selection start point relative to the start of the text. If no selection is defined this function returns the position of the carat.

**6.63.2.8    `size_t CEGUI::Editbox::getSelectionEndIndex (void) const`**

return the current selection end point.

**Returns:**

Index of the selection end point relative to the start of the text. If no selection is defined this function returns the position of the carat.

**6.63.2.9    `size_t CEGUI::Editbox::getSelectionLength (void) const`**

return the length of the current selection (in code points / characters).

**Returns:**

Number of code points (or characters) contained within the currently defined selection.

**6.63.2.10   `utf32 CEGUI::Editbox::getMaskCodePoint (void) const`    `[inline]`**

return the utf32 code point used when rendering masked text.

**Returns:**

utf32 code point value representing the Unicode code point that will be rendered instead of the [Editbox](#) text when rendering in masked mode.

**6.63.2.11   `size_t CEGUI::Editbox::getMaxTextLength (void) const`    `[inline]`**

return the maximum text length set for this [Editbox](#).

**Returns:**

The maximum number of code points (characters) that can be entered into this [Editbox](#).

**Note:**

Depending on the validation string set, the actual length of text that can be entered may be less than the value returned here (it will never be more).

**6.63.2.12   `void CEGUI::Editbox::setReadOnly (bool setting)`**

Specify whether the [Editbox](#) is read-only.

**Parameters:**

*setting* true if the [Editbox](#) is read only and can't be edited by the user, false if the [Editbox](#) is not read only and may be edited by the user.

**Returns:**

Nothing.

**6.63.2.13 void CEGUI::Editbox::setTextMasked (bool *setting*)**

Specify whether the text for the [Editbox](#) will be rendered masked.

**Parameters:**

*setting* true if the [Editbox](#) text should be rendered masked using the currently set mask code point, false if the [Editbox](#) text should be rendered as plain text.

**Returns:**

Nothing.

**6.63.2.14 void CEGUI::Editbox::setValidationString (const String & *validation\_string*)**

Set the text validation string.

**Note:**

Validation is performed by means of a regular expression. If the text matches the regex, the text is said to have passed validation. If the text does not match with the regex then the text fails validation.

**Parameters:**

*validation\_string* [String](#) object containing the validation regex data to be used.

**Returns:**

Nothing.

**6.63.2.15 void CEGUI::Editbox::setCaratIndex (size\_t *carat\_pos*)**

Set the current position of the carat.

**Parameters:**

*carat\_pos* New index for the insert carat relative to the start of the text. If the value specified is greater than the number of characters in the [Editbox](#), the carat is positioned at the end of the text.

**Returns:**

Nothing.

**6.63.2.16 void CEGUI::Editbox::setSelection (size\_t *start\_pos*, size\_t *end\_pos*)**

Define the current selection for the [Editbox](#).

**Parameters:**

*start\_pos* Index of the starting point for the selection. If this value is greater than the number of characters in the [Editbox](#), the selection start will be set to the end of the text.

*end\_pos* Index of the ending point for the selection. If this value is greater than the number of characters in the [Editbox](#), the selection end will be set to the end of the text.

**Returns:**

Nothing.



**6.63.2.17 void CEGUI::Editbox::setMaskCodePoint (utf32 *code\_point*)**

set the utf32 code point used when rendering masked text.

**Parameters:**

*code\_point* utf32 code point value representing the Unicode code point that should be rendered instead of the [Editbox](#) text when rendering in masked mode.

**Returns:**

Nothing.

**6.63.2.18 void CEGUI::Editbox::setMaxTextLength (size\_t *max\_len*)**

set the maximum text length for this [Editbox](#).

**Parameters:**

*max\_len* The maximum number of code points (characters) that can be entered into this [Editbox](#).

**Note:**

Depending on the validation string set, the actual length of text that can be entered may be less than the value set here (it will never be more).

**Returns:**

Nothing.

**6.63.2.19 size\_t CEGUI::Editbox::getTextIndexFromPosition (const Point & *pt*) const**  
[protected]

Return the text code point index that is rendered closest to screen position *pt*.

**Parameters:**

*pt* Point object describing a position on the screen in pixels.

**Returns:**

Code point index into the text that is rendered closest to screen position *pt*.

**6.63.2.20 void CEGUI::Editbox::clearSelection (void)** [protected]

Return the text code point index that is rendered closest to screen position *pt*.

**Parameters:**

*pt* Point object describing a position on the screen in pixels.

**Returns:**

Code point index into the text that is rendered closest to screen position *pt*.

Clear the current selection setting

**6.63.2.21 void CEGUI::Editbox::eraseSelectedText (bool *modify\_text* = true) [protected]**

Erase the currently selected text.

**Parameters:**

*modify\_text* when true, the actual text will be modified. When false, everything is done except erasing the characters.

**6.63.2.22 virtual bool CEGUI::Editbox::testClassName\_impl (const String & *class\_name*) const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.63.2.23 virtual bool CEGUI::Editbox::validateWindowRenderer (const String & *name*) const [inline, protected, virtual]**

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.63.2.24 void CEGUI::Editbox::onTextInvalidatedEvent (WindowEventArgs & *e*) [protected, virtual]**

[Event](#) fired internally when something has caused the current text to now fail validation.

This can be caused by changing the validation string or setting a maximum length that causes the current text to be truncated.

**6.63.2.25 void CEGUI::Editbox::onMouseButtonDown (MouseEventArgs & *e*) [protected, virtual]**

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.63.2.26** void CEGUI::Editbox::onMouseButtonUp (MouseEventArgs & *e*) [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.63.2.27** void CEGUI::Editbox::onMouseDoubleClicked (MouseEventArgs & *e*) [protected, virtual]

Handler called when a mouse button has been double-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.63.2.28** void CEGUI::Editbox::onMouseTripleClicked (MouseEventArgs & *e*) [protected, virtual]

Handler called when a mouse button has been triple-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.63.2.29** void CEGUI::Editbox::onMouseMove (MouseEventArgs & *e*) [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.63.2.30** void CEGUI::Editbox::onCaptureLost (WindowEventArgs & *e*) [protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.63.2.31 void CEGUI::Editbox::onCharacter (KeyEventArgs & e) [protected, virtual]**

Handler called when a character-key has been pressed while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'codepoint' field is set to the Unicode code point (encoded as utf32) for the character typed, and whose 'sysKeys' field represents the combination of SystemKey that were active when the event was generated. All other fields should be considered as 'junk'.

Reimplemented from [CEGUI::Window](#).

**6.63.2.32 void CEGUI::Editbox::onKeyDown (KeyEventArgs & e) [protected, virtual]**

Handler called when a key as been depressed while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'scancode' field is set to the Key::Scan value representing the key that was pressed, and whose 'sysKeys' field represents the combination of SystemKey that were active when the event was generated.

Reimplemented from [CEGUI::Window](#).

**6.63.2.33 void CEGUI::Editbox::onTextChanged (WindowEventArgs & e) [protected, virtual]**

Handler called when the window's text is changed.

**Parameters:**

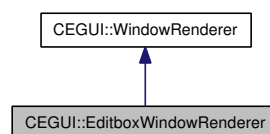
- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

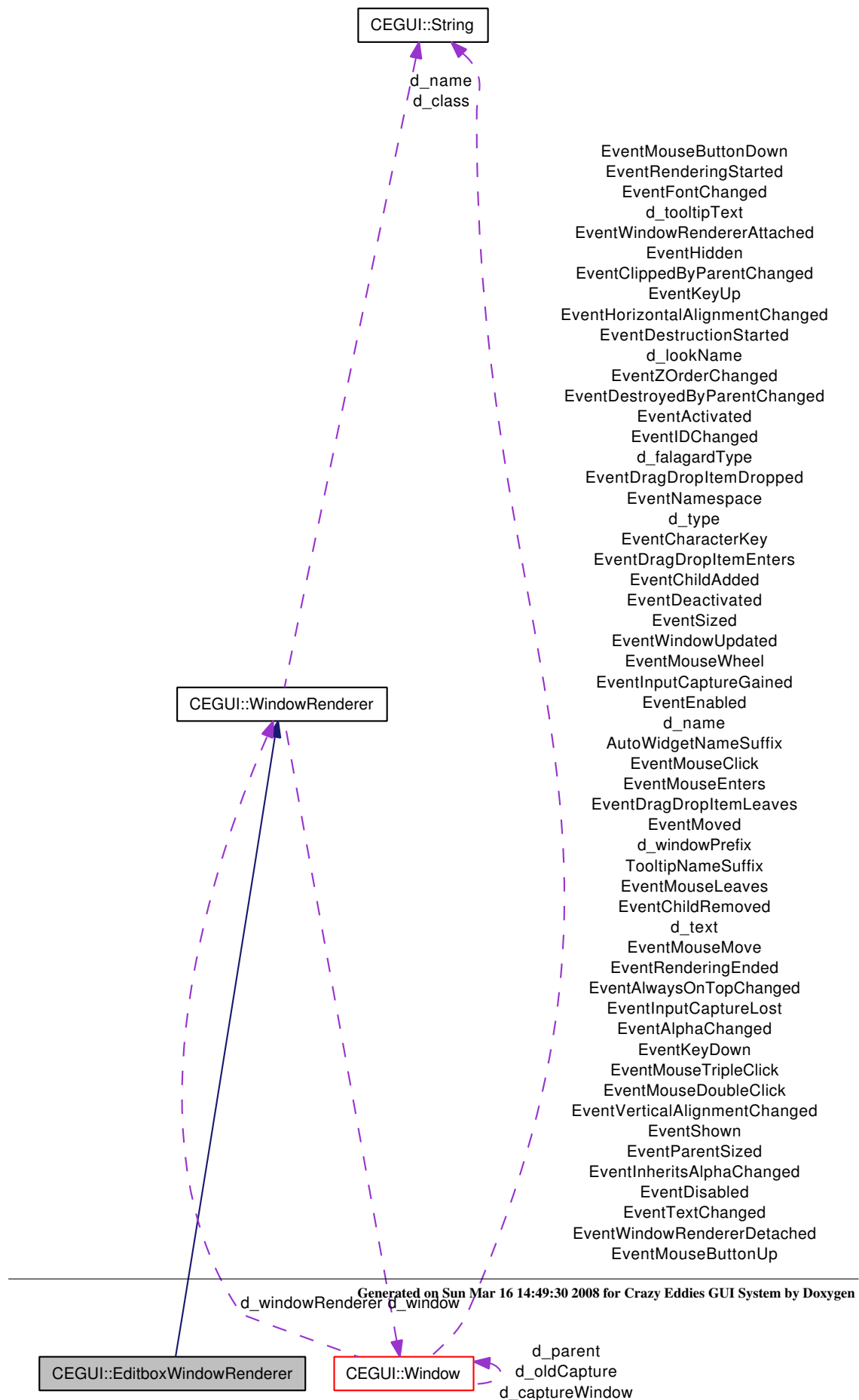
## 6.64 CEGUI::EditboxWindowRenderer Class Reference

Base class for the [EditboxWindowRenderer](#) class.

Inheritance diagram for CEGUI::EditboxWindowRenderer:



Collaboration diagram for CEGUI::EditboxWindowRenderer:



## Public Member Functions

- [EditboxWindowRenderer](#) (const [String](#) &name)  
*Constructor.*
- virtual size\_t [getTextIndexFromPosition](#) (const [Point](#) &pt) const =0  
*Return the text code point index that is rendered closest to screen position pt.*

### 6.64.1 Detailed Description

Base class for the [EditboxWindowRenderer](#) class.

### 6.64.2 Member Function Documentation

#### 6.64.2.1 virtual size\_t CEGUI::EditboxWindowRenderer::getTextIndexFromPosition (const Point &pt) const [pure virtual]

Return the text code point index that is rendered closest to screen position *pt*.

#### Parameters:

*pt* Point object describing a position on the screen in pixels.

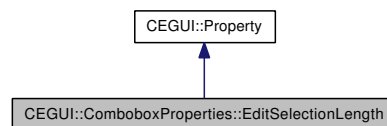
#### Returns:

Code point index into the text that is rendered closest to screen position *pt*.

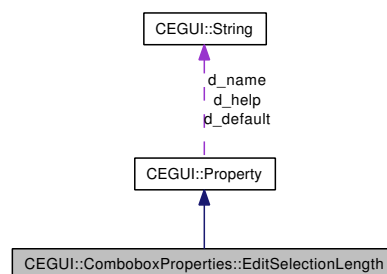
## 6.65 CEGUI::ComboboxProperties::EditSelectionLength Class Reference

[Property](#) to access the current selection length.

Inheritance diagram for CEGUI::ComboboxProperties::EditSelectionLength:



Collaboration diagram for CEGUI::ComboboxProperties::EditSelectionLength:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.65.1 Detailed Description

[Property](#) to access the current selection length.

##### Usage:

- Name: [EditSelectionLength](#)
- Format: "[uint]"

##### Where:

- [uint] is the length of the selection (as a count of the number of code points selected).



## 6.65.2 Member Function Documentation

### 6.65.2.1 String CEGUI::ComboboxProperties::EditSelectionLength::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.65.2.2 void CEGUI::ComboboxProperties::EditSelectionLength::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

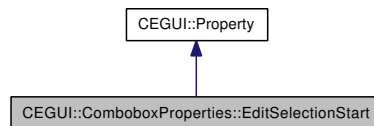
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

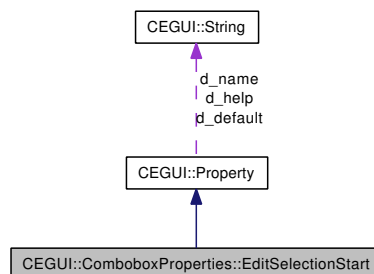
## 6.66 CEGUI::ComboboxProperties::EditSelectionStart Class Reference

[Property](#) to access the current selection start index.

Inheritance diagram for CEGUI::ComboboxProperties::EditSelectionStart:



Collaboration diagram for CEGUI::ComboboxProperties::EditSelectionStart:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.66.1 Detailed Description

[Property](#) to access the current selection start index.

#### Usage:

- Name: [EditSelectionStart](#)
- Format: "[uint]"

#### Where:

- [uint] is the zero based index of the selection start position within the text.

## 6.66.2 Member Function Documentation

### 6.66.2.1 String CEGUI::ComboboxProperties::EditSelectionStart::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.66.2.2 void CEGUI::ComboboxProperties::EditSelectionStart::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

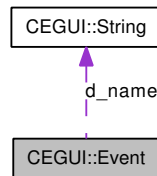
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.67 CEGUI::Event Class Reference

Defines an 'event' which can be subscribed to by interested parties.

Collaboration diagram for CEGUI::Event:



### Public Types

- typedef [RefCounted](#)< [BoundSlot](#) > [Connection](#)  
*Connection object. This is a thin 'smart pointer' wrapper around the actual [BoundSlot](#) that represents the connection. You can use this object to inspect the current connection state and also to disconnect from the event.*
- typedef [CEGUI::SubscriberSlot](#) [Subscriber](#)  
*Subscriber object type. This is now just a typedef to [SubscriberSlot](#), the use of the name [Event::Subscriber](#) is maintained for historical and compatability reasons.*
- typedef unsigned int [Group](#)  
*Type for a subscriber group. You can use the subscriber group to order calls to multiple subscribers. Groups are called in ascending order, with subscribers with no group called last.*

### Public Member Functions

- [Event](#) (const [String](#) &name)  
*Constructs a new [Event](#) object with the specified name.*
- [~Event](#) ()  
*Destructor for [Event](#) objects. Note that this is non-virtual and so you should not sub-class [Event](#).*
- const [String](#) & [getName](#) (void) const  
*Return the name given to this [Event](#) object when it was created.*
- [Connection](#) [subscribe](#) (const [Subscriber](#) &slot)  
*Subscribes some function or object to the [Event](#).*
- [Connection](#) [subscribe](#) ([Group](#) group, const [Subscriber](#) &slot)  
*Subscribes some function or object to the [Event](#).*
- void [operator\(\)](#) ([EventArgs](#) &args)  
*Fires the event. All event subscribers get called in the appropriate sequence.*

## Friends

- void CEGUI::BoundSlot::disconnect ()

## Classes

- class ScopedConnection

*Event::Connection* wrapper that automatically disconnects the connection when the object is deleted (or goes out of scope).

### 6.67.1 Detailed Description

Defines an 'event' which can be subscribed to by interested parties.

An [Event](#) can be subscribed by a function, a member function, or a function object. Whichever option is taken, the function signature needs to be as follows:

```
bool function_name(const EventArgs& args);
```

#### Note:

An [Event](#) object may not be copied.

### 6.67.2 Member Function Documentation

#### 6.67.2.1 const String& CEGUI::Event::getName (void) const [inline]

Return the name given to this [Event](#) object when it was created.

#### Returns:

[String](#) object containing the name of the [Event](#) object.

#### 6.67.2.2 Event::Connection CEGUI::Event::subscribe (const Subscriber & slot)

Subscribes some function or object to the [Event](#).

#### Parameters:

*subscriber* A function, static member function, or function object, with the signature void function\_name(const EventArgs& args). To subscribe a member function you should explicitly create an [Event::Subscriber](#) as this parameter.

#### Returns:

A Connection object which can be used to disconnect (unsubscribe) from the [Event](#), and also to check the connection state.

### 6.67.2.3 **Event::Connection CEGUI::Event::subscribe (Event::Group *group*, const Subscriber & *slot*)**

Subscribes some function or object to the [Event](#).

#### **Parameters:**

*group* The [Event](#) group to subscribe to, subscription groups are called in ascending order, followed by subscriptions with no group. Note that calling order of connections to the same group is unspecified.

*subscriber* A function, static member function, or function object, with the signature void function\_name(const EventArgs& args). To subscribe a member function you should explicitly create an [Event::Subscriber](#) as this parameter.

#### **Returns:**

A Connection object which can be used to disconnect (unsubscribe) from the [Event](#), and also to check the connection state.

### 6.67.2.4 **void CEGUI::Event::operator() (EventArgs & *args*)**

Fires the event. All event subscribers get called in the appropriate sequence.

#### **Parameters:**

*args* An object derived from [EventArgs](#) to be passed to each event subscriber. The 'handled' field will be set to true if any of the called subscribers return that they handled the event.

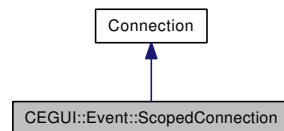
#### **Returns:**

Nothing.

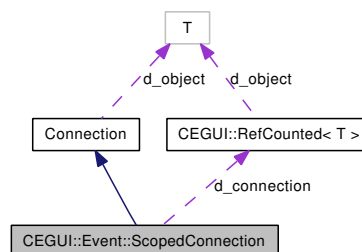
## 6.68 CEGUI::Event::ScopedConnection Class Reference

[Event::Connection](#) wrapper that automatically disconnects the connection when the object is deleted (or goes out of scope).

Inheritance diagram for CEGUI::Event::ScopedConnection:



Collaboration diagram for CEGUI::Event::ScopedConnection:



### Public Member Functions

- **ScopedConnection** (const [Event::Connection](#) &connection)
- [ScopedConnection](#) & **operator=** (const [Event::Connection](#) &connection)
- bool **connected** () const
- void **disconnect** ()

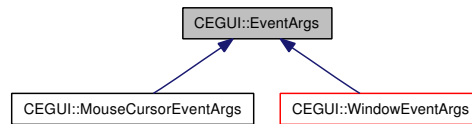
#### 6.68.1 Detailed Description

[Event::Connection](#) wrapper that automatically disconnects the connection when the object is deleted (or goes out of scope).

## 6.69 CEGUI::EventArgs Class Reference

Base class used as the argument to all subscribers [Event](#) object.

Inheritance diagram for CEGUI::EventArgs:



### Public Attributes

- bool [handled](#)  
*handlers should set this to true if they handled the event, or false otherwise.*
- bool [d\\_hasWindow](#)  
*Indicates if this event set has a parent window.*

### 6.69.1 Detailed Description

Base class used as the argument to all subscribers [Event](#) object.

The base [EventArgs](#) class does not contain any useful information, it is intended to be specialised for each type of event that can be generated by objects within the system. The use of this base class allows all event subscribers to have the same function signature.

The *handled* field is used to signal whether an event was actually handled or not. While the event system does not look at this value, code at a higher level can use it to determine how far to propagate an event.



## 6.70 CEGUI::EventSet Class Reference

Class that collects together a set of [Event](#) objects.

Inherited by [CEGUI::GlobalEventSet](#), [CEGUI::MouseCursor](#), [CEGUI::Renderer](#), [CEGUI::System](#), and [CEGUI::Window](#).

### Public Types

- typedef [ConstBaseIterator](#)< EventMap > **Iterator**

### Public Member Functions

- [EventSet](#) ()  
*Constructor for [EventSet](#) objects.*
- virtual [~EventSet](#) (void)  
*Destructor for [EventSet](#) objects.*
- void [addEvent](#) (const [String](#) &name)  
*Add a new [Event](#) to the [EventSet](#) with the given name.*
- void [removeEvent](#) (const [String](#) &name)  
*Removes the [Event](#) with the given name. All connections to the event are disconnected.*
- void [removeAllEvents](#) (void)  
*Remove all [Event](#) objects from the [EventSet](#).*
- bool [isEventPresent](#) (const [String](#) &name)  
*Checks to see if an [Event](#) with the given name is present in the [EventSet](#).*
- virtual [Event::Connection](#) [subscribeEvent](#) (const [String](#) &name, [Event::Subscriber](#) subscriber)  
*Subscribes a handler to the named [Event](#). If the named [Event](#) is not yet present in the [EventSet](#), it is created and added.*
- virtual [Event::Connection](#) [subscribeEvent](#) (const [String](#) &name, [Event::Group](#) group, [Event::Subscriber](#) subscriber)  
*Subscribes a handler to the specified group of the named [Event](#). If the named [Event](#) is not yet present in the [EventSet](#), it is created and added.*
- virtual [Event::Connection](#) [subscribeScriptedEvent](#) (const [String](#) &name, const [String](#) &subscriber\_name)  
*Subscribes the named [Event](#) to a scripted funtion.*
- virtual [Event::Connection](#) [subscribeScriptedEvent](#) (const [String](#) &name, [Event::Group](#) group, const [String](#) &subscriber\_name)  
*Subscribes the specified group of the named [Event](#) to a scripted funtion.*
- virtual void [fireEvent](#) (const [String](#) &name, [EventArgs](#) &args, const [String](#) &eventNamespace="")  
*Fires the named event passing the given [EventArgs](#) object.*

- bool `isMuted` (void) const  
*Return whether the `EventSet` is muted or not.*
- void `setMutedState` (bool setting)  
*Set the mute state for this `EventSet`.*
- `Iterator` `getIterator` (void) const  
*Return a `EventSet::Iterator` object to iterate over the events currently added to the `EventSet`.*

## Protected Types

- typedef std::map< `String`, `Event` \*, `String::FastLessCompare` > `EventMap`

## Protected Member Functions

- `Event` \* `getEventObject` (const `String` &name, bool autoAdd=false)  
*Return a pointer to the `Event` object with the given name, optionally adding such an `Event` object to the `EventSet` if it does not already exist.*
- void `fireEvent_impl` (const `String` &name, `EventArgs` &args)  
*Implementation event firing member.*
- `EventSet` (`EventSet` &e)
- `EventSet` & `operator=` (`EventSet` &e)

## Protected Attributes

- `EventMap` `d_events`
- bool `d_muted`  
*true if events for this `EventSet` have been muted.*

### 6.70.1 Detailed Description

Class that collects together a set of `Event` objects.

The `EventSet` is a means for code to attach a handler function to some named event, and later, for that event to be fired and the subscribed handler(s) called.

As of 0.5, the `EventSet` no longer needs to be filled with available events. Events are now added to the set as they are first used; that is, the first time a handler is subscribed to an event for a given `EventSet`, an `Event` object is created and added to the `EventSet`.

Instead of throwing an exception when firing an event that does not actually exist in the set, we now do nothing (if the `Event` does not exist, then it has no handlers subscribed, and therefore doing nothing is the correct course action).

## 6.70.2 Member Function Documentation

### 6.70.2.1 void CEGUI::EventSet::addEvent (const String & *name*)

Add a new [Event](#) to the [EventSet](#) with the given name.

#### Parameters:

*name* [String](#) object containing the name to give the new [Event](#). The name must be unique for the [EventSet](#).

#### Returns:

Nothing

#### Exceptions:

[AlreadyExistsException](#) Thrown if an [Event](#) already exists named *name*.

### 6.70.2.2 void CEGUI::EventSet::removeEvent (const String & *name*)

Removes the [Event](#) with the given name. All connections to the event are disconnected.

#### Parameters:

*name* [String](#) object containing the name of the [Event](#) to remove. If no such [Event](#) exists, nothing happens.

#### Returns:

Nothing.

### 6.70.2.3 void CEGUI::EventSet::removeAllEvents (void)

Remove all [Event](#) objects from the [EventSet](#).

#### Returns:

Nothing

### 6.70.2.4 bool CEGUI::EventSet::isEventPresent (const String & *name*)

Checks to see if an [Event](#) with the given name is present in the [EventSet](#).

#### Returns:

true if an [Event](#) named *name* was found, or false if the [Event](#) was not found

**6.70.2.5 Event::Connection CEGUI::EventSet::subscribeEvent (const String & *name*, Event::Subscriber *subscriber*)** [virtual]

Subscribes a handler to the named [Event](#). If the named [Event](#) is not yet present in the [EventSet](#), it is created and added.

**Parameters:**

*name* [String](#) object containing the name of the [Event](#) to subscribe to.

*subscriber* Function or object that is to be subscribed to the [Event](#).

**Returns:**

Connection object that can be used to check the status of the [Event](#) connection and to disconnect (unsubscribe) from the [Event](#).

**6.70.2.6 Event::Connection CEGUI::EventSet::subscribeEvent (const String & *name*, Event::Group *group*, Event::Subscriber *subscriber*)** [virtual]

Subscribes a handler to the specified group of the named [Event](#). If the named [Event](#) is not yet present in the [EventSet](#), it is created and added.

**Parameters:**

*name* [String](#) object containing the name of the [Event](#) to subscribe to.

*group* Group which is to be subscribed to. Subscription groups are called in ascending order.

*subscriber* Function or object that is to be subscribed to the [Event](#).

**Returns:**

Connection object that can be used to check the status of the [Event](#) connection and to disconnect (unsubscribe) from the [Event](#).

**6.70.2.7 Event::Connection CEGUI::EventSet::subscribeScriptedEvent (const String & *name*, const String & *subscriber\_name*)** [virtual]

Subscribes the named [Event](#) to a scripted funtion.

**Parameters:**

*name* [String](#) object containing the name of the [Event](#) to subscribe to.

*subscriber\_name* [String](#) object containing the name of the script funtion that is to be subscribed to the [Event](#).

**Returns:**

Connection object that can be used to check the status of the [Event](#) connection and to disconnect (unsubscribe) from the [Event](#).

**6.70.2.8 Event::Connection CEGUI::EventSet::subscribeScriptedEvent (const String & name, Event::Group group, const String & subscriber\_name) [virtual]**

Subscribes the specified group of the named [Event](#) to a scripted funtion.

**Parameters:**

*name* [String](#) object containing the name of the [Event](#) to subscribe to.

*group* Group which is to be subscribed to. Subscription groups are called in ascending order.

*subscriber\_name* [String](#) object containing the name of the script funtion that is to be subscribed to the [Event](#).

**Returns:**

Connection object that can be used to check the status of the [Event](#) connection and to disconnect (unsubscribe) from the [Event](#).

**6.70.2.9 void CEGUI::EventSet::fireEvent (const String & name, EventArgs & args, const String & eventNamespace = "") [virtual]**

Fires the named event passing the given [EventArgs](#) object.

**Parameters:**

*name* [String](#) object holding the name of the [Event](#) that is to be fired (triggered)

*args* The [EventArgs](#) (or derived) object that is to be bassed to each subscriber of the [Event](#). Once all subscribers have been called the 'handled' field of the event is updated appropriately.

*eventNamespace* [String](#) object describing the global event namespace prefix for this event.

**Returns:**

Nothing.

Reimplemented in [CEGUI::GlobalEventSet](#).

**6.70.2.10 bool CEGUI::EventSet::isMuted (void) const**

Return whether the [EventSet](#) is muted or not.

**Returns:**

- true if the [EventSet](#) is muted. All requests to fire events will be ignored.
- false if the [EventSet](#) is not muted. All requests to fire events are processed as normal.

**6.70.2.11 void CEGUI::EventSet::setMutedState (bool setting)**

Set the mute state for this [EventSet](#).

**Parameters:**

*setting* • true if the [EventSet](#) is to be muted (no further event firing requests will be honoured until [EventSet](#) is unmuted).

- false if the [EventSet](#) is not to be muted and all events should fired as requested.

**Returns:**

Nothing.

**6.70.2.12** `Event * CEGUI::EventSet::getEventObject (const String & name, bool autoAdd = false)` [protected]

Return a pointer to the [Event](#) object with the given name, optionally adding such an [Event](#) object to the [EventSet](#) if it does not already exist.

**Parameters:**

*name* [String](#) object holding the name of the [Event](#) to return.

- autoAdd*
- true if an [Event](#) object named *name* should be added to the set if such an [Event](#) does not currently exist.
  - false if no object should automatically be added to the set. In this case, if the [Event](#) does not already exist 0 will be returned.

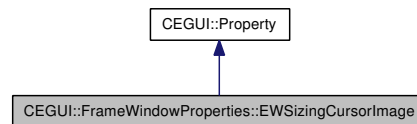
**Returns:**

Pointer to the [Event](#) object in this [EventSet](#) with the specified name. Or 0 if such an [Event](#) does not exist and *autoAdd* was false.

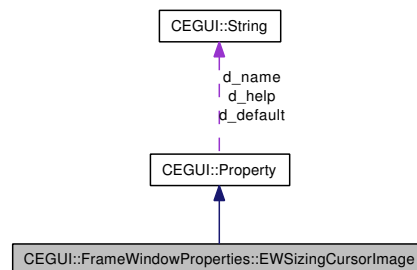
## 6.71 CEGUI::FrameWindowProperties::EWSizingCursorImage Class Reference

[Property](#) to access the E-W (left/right) sizing cursor image.

Inheritance diagram for CEGUI::FrameWindowProperties::EWSizingCursorImage:



Collaboration diagram for CEGUI::FrameWindowProperties::EWSizingCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

### 6.71.1 Detailed Description

[Property](#) to access the E-W (left/right) sizing cursor image.

Usage:

- Name: [EWSizingCursorImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.71.2 Member Function Documentation

#### 6.71.2.1 String CEGUI::FrameWindowProperties::EWSizingCursorImage::get (const [PropertyReceiver](#) \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.71.2.2** `void CEGUI::FrameWindowProperties::EWSizingCursorImage::set (PropertyReceiver *  
receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

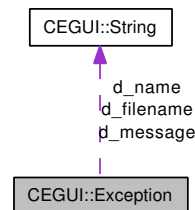


## 6.72 CEGUI::Exception Class Reference

Root exception class used within the GUI system.

Inherited by [CEGUI::AlreadyExistsException](#), [CEGUI::FileIOException](#), [CEGUI::GenericException](#), [CEGUI::InvalidRequestException](#), [CEGUI::MemoryException](#), [CEGUI::NullObjectException](#), [CEGUI::ObjectInUseException](#), [CEGUI::RendererException](#), [CEGUI::ScriptException](#), and [CEGUI::UnknownObjectException](#).

Collaboration diagram for CEGUI::Exception:



### Public Member Functions

- virtual [~Exception](#) (void)  
*Virtual destructor.*
- const [String](#) & [getMessage](#) (void) const  
*Return a reference to the [String](#) object describing the reason for the exception being thrown.*
- const [String](#) & [getName](#) () const  
*Return a reference to the [String](#) object containing the exception name (i.e. class type).*
- const [String](#) & [getFileName](#) (void) const  
*Return a reference to the [String](#) object containing the name of the file where the exception occurred.*
- const int [getLine](#) (void) const  
*Return the line number where the exception occurred.*

### Protected Member Functions

- [Exception](#) (const [String](#) &message="", const [String](#) &name="CEGUI::Exception", const [String](#) &filename="", int line=0)  
*Protected constructor that prevents instantiations (users should employ derived exception classes instead) and that is responsible for logging the exception.*

### Protected Attributes

- [String](#) [d\\_message](#)  
*Holds the reason for the exception being thrown.*

- [String d\\_filename](#)

*Holds the name of the file where the exception occurred.*

- [String d\\_name](#)

*Holds the class name of the exception (e.g. [CEGUI::ObjectInUseException](#)).*

- [int d\\_line](#)

*Holds the line number where the exception occurred.*

## 6.72.1 Detailed Description

Root exception class used within the GUI system.

## 6.72.2 Constructor & Destructor Documentation

### 6.72.2.1 `CEGUI::Exception::Exception (const String & message = "", const String & name = "CEGUI::Exception", const String & filename = "", int line = 0) [protected]`

Protected constructor that prevents instantiations (users should employ derived exception classes instead) and that is responsible for logging the exception.

#### Parameters:

*message* [String](#) object describing the reason for the exception being thrown.

*name* [String](#) object describing the exception class name (e.g. [CEGUI::UnknownObjectException](#))

*filename* [String](#) object containing the name of the file where the exception occurred.

*line* Integer representing the line number where the exception occurred.

## 6.72.3 Member Function Documentation

### 6.72.3.1 `const String& CEGUI::Exception::getMessage (void) const [inline]`

Return a reference to the [String](#) object describing the reason for the exception being thrown.

#### Returns:

[String](#) object containing a message describing the reason for the exception.

### 6.72.3.2 `const String& CEGUI::Exception::getName () const [inline]`

Return a reference to the [String](#) object containing the exception name (i.e. class type).

#### Returns:

[String](#) object containing the exception name.

**6.72.3.3 const String& CEGUI::Exception::getFileName (void) const** [inline]

Return a reference to the [String](#) object containing the name of the file where the exception occurred.

**Returns:**

[String](#) object containing the name of the file where the exception occurred.

**6.72.3.4 const int CEGUI::Exception::getLine (void) const** [inline]

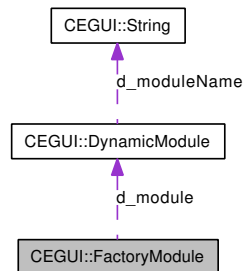
Return the line number where the exception occurred.

Integer representing the line number where the exception occurred.

## 6.73 CEGUI::FactoryModule Class Reference

Class that encapsulates access to a dynamic loadable module containing implementations of Windows, Widgets, and their factories.

Collaboration diagram for CEGUI::FactoryModule:



### Public Member Functions

- **FactoryModule** (const [String](#) &filename)  
*Construct the [FactoryModule](#) object by loading the dynamic loadable module specified.*
- virtual **~FactoryModule** (void)  
*Destroys the [FactoryModule](#) object and unloads any loadable module.*
- void **registerFactory** (const [String](#) &type) const  
*Register a [WindowFactory](#) for type Windows.*
- uint **registerAllFactories** () const  
*Register all factories available in this module.*

### 6.73.1 Detailed Description

Class that encapsulates access to a dynamic loadable module containing implementations of Windows, Widgets, and their factories.

### 6.73.2 Constructor & Destructor Documentation

#### 6.73.2.1 CEGUI::FactoryModule::FactoryModule (const [String](#) &filename)

Construct the [FactoryModule](#) object by loading the dynamic loadable module specified.

#### Parameters:

*filename* [String](#) object holding the filename of a loadable module.

#### Returns:

Nothing

### 6.73.2.2 CEGUI::FactoryModule::~~FactoryModule (void) [virtual]

Destroys the [FactoryModule](#) object and unloads any loadable module.

**Returns:**

Nothing

## 6.73.3 Member Function Documentation

### 6.73.3.1 void CEGUI::FactoryModule::registerFactory (const String & *type*) const

Register a [WindowFactory](#) for *type* Windows.

**Parameters:**

*type* [String](#) object holding the name of the [Window](#) type a factory is to be registered for.

**Returns:**

Nothing.

### 6.73.3.2 uint CEGUI::FactoryModule::registerAllFactories () const

Register all factories available in this module.

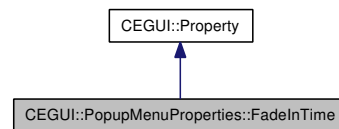
**Returns:**

uint value indicating the number of factories registered.

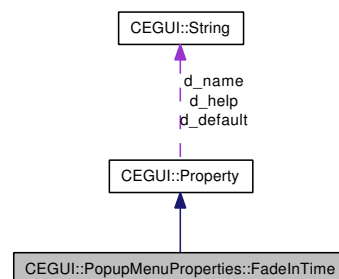
## 6.74 CEGUI::PopupMenuProperties::FadeInTime Class Reference

[Property](#) to access the fade in time in seconds of the popup menu.

Inheritance diagram for CEGUI::PopupMenuProperties::FadeInTime:



Collaboration diagram for CEGUI::PopupMenuProperties::FadeInTime:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.74.1 Detailed Description

[Property](#) to access the fade in time in seconds of the popup menu.

##### Usage:

- Name: [FadeInTime](#)
- Format: "[float]".

##### Where:

- [float] represents the fade in time in seconds of the popup menu.

## 6.74.2 Member Function Documentation

### 6.74.2.1 String CEGUI::PopupMenuProperties::FadeInTime::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.74.2.2 void CEGUI::PopupMenuProperties::FadeInTime::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

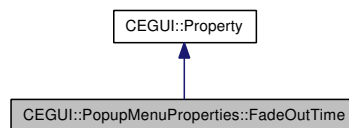
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

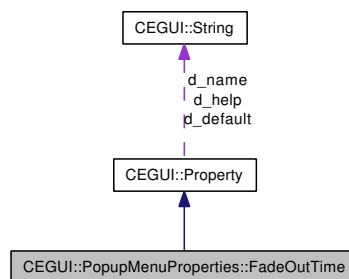
## 6.75 CEGUI::PopupMenuProperties::FadeOutTime Class Reference

[Property](#) to access the fade out time in seconds of the popup menu.

Inheritance diagram for CEGUI::PopupMenuProperties::FadeOutTime:



Collaboration diagram for CEGUI::PopupMenuProperties::FadeOutTime:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.75.1 Detailed Description

[Property](#) to access the fade out time in seconds of the popup menu.

##### Usage:

- Name: [FadeOutTime](#)
- Format: "[float]".

##### Where:

- [float] represents the fade out time in seconds of the popup menu.



## 6.75.2 Member Function Documentation

### 6.75.2.1 String CEGUI::PopupMenuProperties::FadeOutTime::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.75.2.2 void CEGUI::PopupMenuProperties::FadeOutTime::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

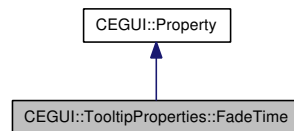
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

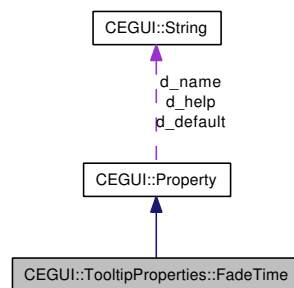
## 6.76 CEGUI::TooltipProperties::FadeTime Class Reference

[Property](#) to access the duration of the fade effect for the tooltip.

Inheritance diagram for CEGUI::TooltipProperties::FadeTime:



Collaboration diagram for CEGUI::TooltipProperties::FadeTime:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.76.1 Detailed Description

[Property](#) to access the duration of the fade effect for the tooltip.

#### Usage:

- Name: [FadeTime](#)
- Format: "[float]".

#### Where:

- [float] specifies the number of seconds over which the fade in / fade out effect will happen.

## 6.76.2 Member Function Documentation

### 6.76.2.1 String CEGUI::TooltipProperties::FadeTime::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.76.2.2 void CEGUI::TooltipProperties::FadeTime::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

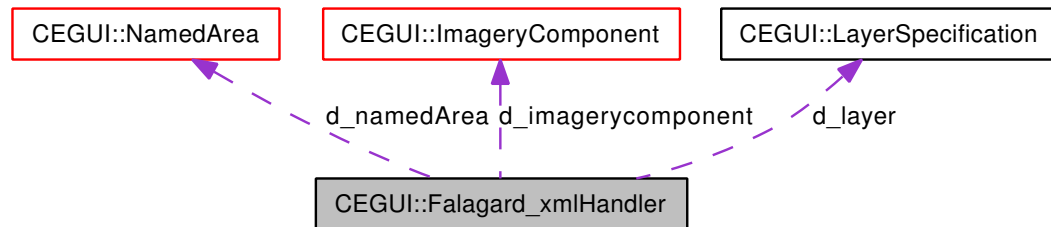
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.77 CEGUI::Falagard\_xmlHandler Class Reference

Handler class used to parse look & feel XML files used by the Falagard system.

Collaboration diagram for CEGUI::Falagard\_xmlHandler:



### Public Member Functions

- [Falagard\\_xmlHandler](#) ([WidgetLookManager](#) \*mgr)  
*Constructor for [Falagard\\_xmlHandler](#) objects.*
- [~Falagard\\_xmlHandler](#) ()  
*Destructor for [Falagard\\_xmlHandler](#) objects.*
- void **elementStart** (const [String](#) &element, const [XMLAttributes](#) &attributes)
- void **elementEnd** (const [String](#) &element)

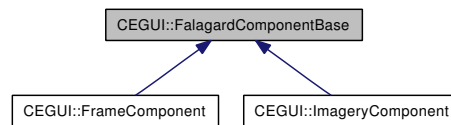
### 6.77.1 Detailed Description

Handler class used to parse look & feel XML files used by the Falagard system.

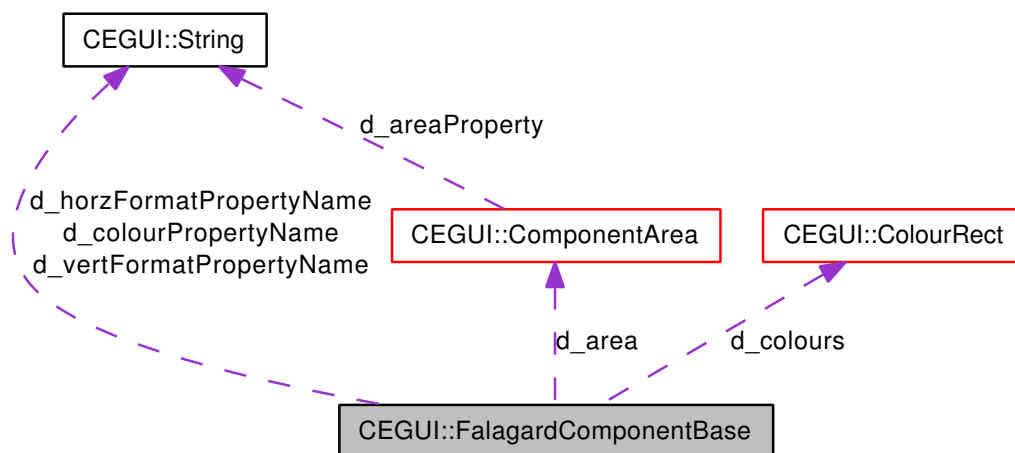
## 6.78 CEGUI::FalagardComponentBase Class Reference

Common base class used for renderable components within an [ImagerySection](#).

Inheritance diagram for CEGUI::FalagardComponentBase:



Collaboration diagram for CEGUI::FalagardComponentBase:



### Public Member Functions

- [FalagardComponentBase](#) ()  
*Constructor.*
- virtual [~FalagardComponentBase](#) ()  
*Destructor.*
- void [render](#) ([Window](#) &srcWindow, float base\_z, const [CEGUI::ColourRect](#) \*modColours=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const  
*Render this component. More correctly, the component is cached for rendering.*
- void [render](#) ([Window](#) &srcWindow, const [Rect](#) &baseRect, float base\_z, const [CEGUI::ColourRect](#) \*modColours=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const  
*Render this component. More correctly, the component is cached for rendering.*
- const [ComponentArea](#) & [getComponentArea](#) () const  
*Return the [ComponentArea](#) of this [ImageryComponent](#).*
- void [setComponentArea](#) (const [ComponentArea](#) &area)  
*Set the [ImageryComponent](#)'s [ComponentArea](#).*

- const [ColourRect](#) & [getColours](#) () const  
*Return the [ColourRect](#) set for use by this [ImageryComponent](#).*
- void [setColours](#) (const [ColourRect](#) &cols)  
*Set the colours to be used by this [ImageryComponent](#).*
- void [setColoursPropertySource](#) (const [String](#) &property)  
*Set the name of the property where [colour](#) values can be obtained.*
- void [setColoursPropertyIsColourRect](#) (bool setting=true)  
*Set whether the colours property source represents a full [ColourRect](#).*
- void [setVertFormattingPropertySource](#) (const [String](#) &property)  
*Set the name of the property where vertical formatting option can be obtained.*
- void [setHorzFormattingPropertySource](#) (const [String](#) &property)  
*Set the name of the property where horizontal formatting option can be obtained.*

## Protected Member Functions

- void [initColoursRect](#) (const [Window](#) &wnd, const [ColourRect](#) \*modCols, [ColourRect](#) &cr) const  
*Helper method to initialise a [ColourRect](#) with appropriate values according to the way the [ImageryComponent](#) is set up.*
- virtual void [render\\_impl](#) ([Window](#) &srcWindow, [Rect](#) &destRect, float base\_z, const [CEGUI::ColourRect](#) \*modColours, const [Rect](#) \*clipper, bool clipToDisplay) const =0  
*Method to do main render caching work.*
- bool [writeColoursXML](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes xml for the colours to a [OutStream](#). Will prefer property colours before explicit.*
- bool [writeVertFormatXML](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes xml for the vertical formatting to a [OutStream](#) if such a property is defined.*
- bool [writeHorzFormatXML](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes xml for the horizontal formatting to a [OutStream](#) if such a property is defined.*

## Protected Attributes

- [ComponentArea d\\_area](#)  
*Destination area for this component.*
- [ColourRect d\\_colours](#)  
*base colours to be applied when rendering the image component.*
- [String d\\_colourPropertyName](#)

*name of property to fetch colours from.*

- `bool d_colourPropertyIsRect`  
*true if the `colour` property will fetch a full `ColourRect`.*
- `String d_vertFormatPropertyName`  
*name of property to fetch vertical formatting setting from.*
- `String d_horzFormatPropertyName`  
*name of property to fetch horizontal formatting setting from.*

## 6.78.1 Detailed Description

Common base class used for renderable components within an [ImagerySection](#).

## 6.78.2 Member Function Documentation

**6.78.2.1** `void CEGUI::FalagardComponentBase::render (Window & srcWindow, float base_z, const CEGUI::ColourRect * modColours = 0, const Rect * clipper = 0, bool clipToDisplay = false) const`

Render this component. More correctly, the component is cached for rendering.

### Parameters:

*srcWindow* [Window](#) to use as the base for translating the component's [ComponentArea](#) into pixel values.

*base\_z* The z value to use for rendering the component. Note that this is not the final z value to use, but some z offset from a currently unknown starting value.

*modColours* [ColourRect](#) describing colours that are to be modulated with the component's stored [colour](#) values to calculate a set of 'final' [colour](#) values to be used. May be 0.

### Returns:

Nothing.

**6.78.2.2** `void CEGUI::FalagardComponentBase::render (Window & srcWindow, const Rect & baseRect, float base_z, const CEGUI::ColourRect * modColours = 0, const Rect * clipper = 0, bool clipToDisplay = false) const`

Render this component. More correctly, the component is cached for rendering.

### Parameters:

*srcWindow* [Window](#) to use as the base for translating the component's [ComponentArea](#) into pixel values.

*baseRect* [Rect](#) to use as the base for translating the component's [ComponentArea](#) into pixel values.

*base\_z* The z value to use for rendering the component. Note that this is not the final z value to use, but some z offset from a currently unknown starting value.

*modColours* [ColourRect](#) describing colours that are to be modulated with the component's stored [colour](#) values to calculate a set of 'final' [colour](#) values to be used. May be 0.

**Returns:**

Nothing.

**6.78.2.3** `const ComponentArea & CEGUI::FalagardComponentBase::getComponentArea () const`

Return the [ComponentArea](#) of this [ImageryComponent](#).

**Returns:**

[ComponentArea](#) object describing the [ImageryComponent](#)'s current target area.

**6.78.2.4** `void CEGUI::FalagardComponentBase::setComponentArea (const ComponentArea & area)`

Set the [ImageryComponent](#)'s [ComponentArea](#).

**Parameters:**

*area* [ComponentArea](#) object describing a new target area for the [ImageryComponent](#).

**Returns:**

Nothing.

**6.78.2.5** `const ColourRect & CEGUI::FalagardComponentBase::getColours () const`

Return the [ColourRect](#) set for use by this [ImageryComponent](#).

**Returns:**

[ColourRect](#) object holding the colours currently in use by this [ImageryComponent](#).

**6.78.2.6** `void CEGUI::FalagardComponentBase::setColours (const ColourRect & cols)`

Set the colours to be used by this [ImageryComponent](#).

**Parameters:**

*cols* [ColourRect](#) object describing the colours to be used by this [ImageryComponent](#).



**6.78.2.7 void CEGUI::FalagardComponentBase::setColoursPropertySource (const String & *property*)**

Set the name of the property where [colour](#) values can be obtained.

**Parameters:**

*property* [String](#) containing the name of the property.

**Returns:**

Nothing.

**6.78.2.8 void CEGUI::FalagardComponentBase::setColoursPropertyIsColourRect (bool *setting* = true)**

Set whether the colours property source represents a full [ColourRect](#).

**Parameters:**

- setting* • true if the colours property will access a [ColourRect](#) object.
- false if the colours property will access a [colour](#) object.

**Returns:**

Nothing.

**6.78.2.9 void CEGUI::FalagardComponentBase::setVertFormattingPropertySource (const String & *property*)**

Set the name of the property where vertical formatting option can be obtained.

**Parameters:**

*property* [String](#) containing the name of the property.

**Returns:**

Nothing.

**6.78.2.10 void CEGUI::FalagardComponentBase::setHorzFormattingPropertySource (const String & *property*)**

Set the name of the property where horizontal formatting option can be obtained.

**Parameters:**

*property* [String](#) containing the name of the property.

**Returns:**

Nothing.

**6.78.2.11 void CEGUI::FalagardComponentBase::initColoursRect (const Window & *wnd*, const ColourRect \* *modCols*, ColourRect & *cr*) const** [protected]

Helper method to initialise a [ColourRect](#) with appropriate values according to the way the [ImageryComponent](#) is set up.

This will try and get values from multiple places:

- a property attached to *wnd*
- or the integral *d\_colours* value.

**6.78.2.12 bool CEGUI::FalagardComponentBase::writeColoursXML (XMLSerializer & *xml\_stream*) const** [protected]

Writes xml for the colours to a OutStream. Will prefer property colours before explicit.

**Note:**

This is intended as a helper method for sub-classes when outputting xml to a stream.

**Returns:**

- true if xml element was written.
- false if nothing was output due to the formatting not being set (sub-class may then choose to do something else.)

**6.78.2.13 bool CEGUI::FalagardComponentBase::writeVertFormatXML (XMLSerializer & *xml\_stream*) const** [protected]

Writes xml for the vertical formatting to a OutStream if such a property is defined.

**Note:**

This is intended as a helper method for sub-classes when outputting xml to a stream.

**Returns:**

- true if xml element was written.
- false if nothing was output due to the formatting not being set (sub-class may then choose to do something else.)

**6.78.2.14 bool CEGUI::FalagardComponentBase::writeHorzFormatXML (XMLSerializer & *xml\_stream*) const** [protected]

Writes xml for the horizontal formatting to a OutStream if such a property is defined.

**Note:**

This is intended as a helper method for sub-classes when outputting xml to a stream.

**Returns:**

- true if xml element was written.
- false if nothing was output due to the formatting not being set (sub-class may then choose to do something else.)

## 6.79 CEGUI::FalagardXMLHelper Class Reference

Utility helper class primarily intended for use by the falagard xml parser.

### Static Public Member Functions

- static [VerticalFormatting](#) **stringToVertFormat** (const [String](#) &str)
- static [HorizontalFormatting](#) **stringToHorzFormat** (const [String](#) &str)
- static [VerticalAlignment](#) **stringToVertAlignment** (const [String](#) &str)
- static [HorizontalAlignment](#) **stringToHorzAlignment** (const [String](#) &str)
- static [DimensionType](#) **stringToDimensionType** (const [String](#) &str)
- static [VerticalTextFormatting](#) **stringToVertTextFormat** (const [String](#) &str)
- static [HorizontalTextFormatting](#) **stringToHorzTextFormat** (const [String](#) &str)
- static [FontMetricType](#) **stringToFontMetricType** (const [String](#) &str)
- static [DimensionOperator](#) **stringToDimensionOperator** (const [String](#) &str)
- static [FrameImageComponent](#) **stringToFrameImageComponent** (const [String](#) &str)
- static [String](#) **vertFormatToString** ([VerticalFormatting](#) format)
- static [String](#) **horzFormatToString** ([HorizontalFormatting](#) format)
- static [String](#) **vertAlignmentToString** ([VerticalAlignment](#) alignment)
- static [String](#) **horzAlignmentToString** ([HorizontalAlignment](#) alignment)
- static [String](#) **dimensionTypeToString** ([DimensionType](#) dim)
- static [String](#) **vertTextFormatToString** ([VerticalTextFormatting](#) format)
- static [String](#) **horzTextFormatToString** ([HorizontalTextFormatting](#) format)
- static [String](#) **fontMetricTypeToString** ([FontMetricType](#) metric)
- static [String](#) **dimensionOperatorToString** ([DimensionOperator](#) op)
- static [String](#) **frameImageComponentToString** ([FrameImageComponent](#) imageComp)

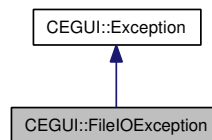
### 6.79.1 Detailed Description

Utility helper class primarily intended for use by the falagard xml parser.

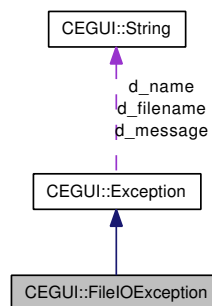
## 6.80 CEGUI::FileIOException Class Reference

[Exception](#) class used when a file handling problem occurs.

Inheritance diagram for CEGUI::FileIOException:



Collaboration diagram for CEGUI::FileIOException:



### Public Member Functions

- [FileIOException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the file IO exception by calling the base class.*

### 6.80.1 Detailed Description

[Exception](#) class used when a file handling problem occurs.

### 6.80.2 Constructor & Destructor Documentation

- 6.80.2.1** CEGUI::FileIOException::FileIOException (const [String](#) & *message*, const [String](#) & *file* = "unknown", int *line* = 0) [inline]

Constructor that is responsible for logging the file IO exception by calling the base class.

#### Parameters:

- message* [String](#) object describing the reason for the file IO exception being thrown.
- filename* [String](#) object containing the name of the file where the file IO exception occurred.
- line* Integer representing the line number where the file IO exception occurred.

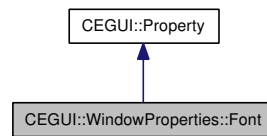
**Remarks:**

The file IO exception name is automatically passed to the base class as "CEGUI::FileIOException".

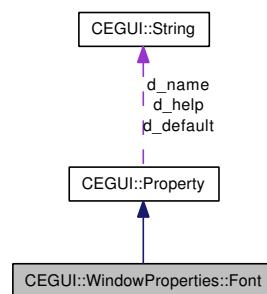
## 6.81 CEGUI::WindowProperties::Font Class Reference

[Property](#) to access window [Font](#) setting.

Inheritance diagram for CEGUI::WindowProperties::Font:



Collaboration diagram for CEGUI::WindowProperties::Font:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- bool **isDefault** (const [PropertyReceiver](#) \*receiver) const  
*Returns whether the property is at it's default value.*

### 6.81.1 Detailed Description

[Property](#) to access window [Font](#) setting.

This property offers access to the current [Font](#) setting for the window.

#### Usage:

- Name: [Font](#)
- Format: "[text]".

#### Where:

- [text] is the name of the [Font](#) to assign for this window. The [Font](#) specified must already be loaded.

## 6.81.2 Member Function Documentation

### 6.81.2.1 String CEGUI::WindowProperties::Font::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.81.2.2 void CEGUI::WindowProperties::Font::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

### 6.81.2.3 bool CEGUI::WindowProperties::Font::isDefault (const PropertyReceiver \* *receiver*) const [virtual]

Returns whether the property is at it's default value.

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

- true if the property has it's default value.
- false if the property has been modified from it's default value.

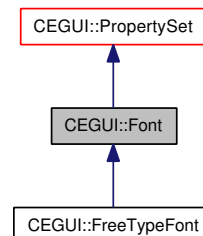
Reimplemented from [CEGUI::Property](#).



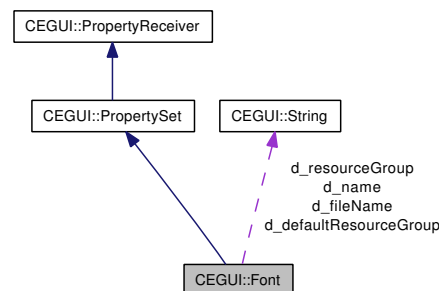
## 6.82 CEGUI::Font Class Reference

Class that encapsulates text rendering functionality for a typeface.

Inheritance diagram for CEGUI::Font:



Collaboration diagram for CEGUI::Font:



### Public Member Functions

- virtual void [load](#) ()=0

*Complete font loading. If you create the font from an XML file, this method is invoked automatically after reading all the required data from the [XMLAttributes](#) object. If you create the font manually, it is your responsibility to call this function as soon as you set up all the appropriate fields of the [Font](#) object.*

- bool [isCodepointAvailable](#) (utf32 cp) const

*Return whether this [Font](#) can draw the specified code-point.*

- size\_t [drawText](#) (const [String](#) &text, const [Rect](#) &draw\_area, float z, const [Rect](#) &clip\_rect, [TextFormatting](#) fmt, const [ColourRect](#) &colours, float x\_scale=1.0f, float y\_scale=1.0f)

*Draw text into a specified area of the display.*

- size\_t [drawText](#) (const [String](#) &text, const [Rect](#) &draw\_area, float z, const [Rect](#) &clip\_rect, [TextFormatting](#) fmt, float x\_scale=1.0f, float y\_scale=1.0f)

*Draw text into a specified area of the display using default colours.*

- void [drawText](#) (const [String](#) &text, const [Rect](#) &draw\_area, float z, const [Rect](#) &clip\_rect, float x\_scale=1.0f, float y\_scale=1.0f)

*Draw text into a specified area of the display with default colours and default formatting ([LeftAligned](#)).*

- `size_t drawText` (const `String` &text, const `Rect` &draw\_area, float z, `TextFormatting` fmt, const `ColourRect` &colours, float x\_scale=1.0f, float y\_scale=1.0f)  
*Draw text into a specified area of the display.*
- `size_t drawText` (const `String` &text, const `Rect` &draw\_area, float z, `TextFormatting` fmt, float x\_scale=1.0f, float y\_scale=1.0f)  
*Draw text into a specified area of the display with default colours.*
- `void drawText` (const `String` &text, const `Rect` &draw\_area, float z, float x\_scale=1.0f, float y\_scale=1.0f)  
*Draw text into a specified area of the display with default colours and default formatting (LeftAligned).*
- `void drawText` (const `String` &text, const `Vector3` &position, const `Rect` &clip\_rect, const `ColourRect` &colours, float x\_scale=1.0f, float y\_scale=1.0f)  
*Draw text at the specified location.*
- `void drawText` (const `String` &text, const `Vector3` &position, const `Rect` &clip\_rect, float x\_scale=1.0f, float y\_scale=1.0f)  
*Draw text at the specified location with default colours.*
- `virtual void setNativeResolution` (const `Size` &size)  
*Set the native resolution for this `Font`.*
- `virtual void notifyScreenResolution` (const `Size` &size)  
*Notify the `Font` of the current (usually new) display resolution.*
- `float getLineSpacing` (float y\_scale=1.0f) const  
*Return the pixel line spacing value for.*
- `float getFontHeight` (float y\_scale=1.0f) const  
*return the exact pixel height of the font.*
- `float getBaseline` (float y\_scale=1.0f) const  
*Return the number of pixels from the top of the highest glyph to the baseline.*
- `float getTextExtent` (const `String` &text, float x\_scale=1.0f)  
*Return the pixel width of the specified text if rendered with this `Font`.*
- `size_t getCharAtPixel` (const `String` &text, float pixel, float x\_scale=1.0f)  
*Return the index of the closest text character in `String` text that corresponds to pixel location pixel if the text were rendered.*
- `size_t getCharAtPixel` (const `String` &text, size\_t start\_char, float pixel, float x\_scale=1.0f)  
*Return the index of the closest text character in `String` text, starting at character index start\_char, that corresponds to pixel location pixel if the text were to be rendered.*
- `size_t getFormattedLineCount` (const `String` &text, const `Rect` &format\_area, `TextFormatting` fmt, float x\_scale=1.0f)  
*Return the number of lines the given text would be formatted to.*

- float `getFormattedTextExtent` (const [String](#) &text, const [Rect](#) &format\_area, [TextFormatting](#) fmt, float x\_scale=1.0f)

*Return the horizontal pixel extent given text would be formatted to.*

## Static Public Member Functions

- static void `setDefaultResourceGroup` (const [String](#) &resourceGroup)

*Sets the default resource group to be used when loading font data.*

- static const [String](#) & `getDefaultResourceGroup` ()

*Returns the default resource group currently set for Fonts.*

## Static Public Attributes

- static const [argb\\_t](#) `DefaultColour` = 0xFFFFFFFF

*Colour value used whenever a [colour](#) is not specified.*

## Protected Types

- typedef std::map< utf32, [FontGlyph](#) > `CodepointMap`

## Protected Member Functions

- [Font](#) (const [String](#) &name, const [String](#) &fontname, const [String](#) &resourceGroup="")

*Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to set up all remaining font parameters after constructing the [Font](#) object, and finally calling the `load()` method which will make font available for use. All font parameters that are not initialized are set to sensible default values.*

- [Font](#) (const [XMLAttributes](#) &attributes)

*Constructs a new [Font](#) object and instantly loads it. The font is ready for use right after creation, there is no need to `load()` it. All data required by this font is loaded from the provided [XMLAttributes](#) object.*

- virtual `~Font` ()

*Destroys a [Font](#) object.*

- virtual void `defineMapping` (const [XMLAttributes](#) &attributes)

*Define a glyph mapping (handle a <Mapping> XML element).*

- virtual void `updateFont` ()=0

*Update the font as required according to the current parameters.*

- size\_t `drawWrappedText` (const [String](#) &text, const [Rect](#) &draw\_area, float z, const [Rect](#) &clip\_rect, [TextFormatting](#) fmt, const [ColourRect](#) &colours, float x\_scale=1.0f, float y\_scale=1.0f)

*draws wrapped text. returns number of lines output.*

- size\_t `getNextWord` (const [String](#) &in\_string, size\_t start\_idx, [String](#) &out\_string) const

*helper function for renderWrappedText to get next word of a string*

- void [drawTextLine](#) (const [String](#) &text, const [Vector3](#) &position, const [Rect](#) &clip\_rect, const [ColourRect](#) &colours, float x\_scale=1.0f, float y\_scale=1.0f)

*Draw a line of text. No formatting is applied.*

- void [drawTextLineJustified](#) (const [String](#) &text, const [Rect](#) &draw\_area, const [Vector3](#) &position, const [Rect](#) &clip\_rect, const [ColourRect](#) &colours, float x\_scale=1.0f, float y\_scale=1.0f)

*Draw a justified line of text.*

- float [getWrappedTextExtent](#) (const [String](#) &text, float wrapWidth, float x\_scale=1.0f)

*returns extent of widest line of wrapped text.*

- const [FontGlyph](#) \* [getGlyphData](#) (utf32 codepoint)

*Return a pointer to the glyphDat struct for the given codepoint, or 0 if the codepoint does not have a glyph defined.*

- void [setMaxCodepoint](#) (utf32 codepoint)

*Set the maximal glyph index. This reserves the respective number of bits in the d\_glyphPageLoaded array.*

- virtual void [rasterize](#) (utf32 start\_codepoint, utf32 end\_codepoint)

*This function prepares a certain range of glyphs to be ready for displaying. This means that after returning from this function glyphs from d\_cp\_map[start\_codepoint] to d\_cp\_map[end\_codepoint] should have their d\_image member set. If there is an error during rasterization of some glyph, it's okay to leave the d\_image field set to NULL, in which case such glyphs will be skipped from display.*

- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const

*Writes an xml representation of this [Font](#) to out\_stream.*

- virtual void [writeXMLToStream\\_impl](#) ([XMLSerializer](#) &xml\_stream) const =0

*Same as [writeXMLToStream\(\)](#) but called from inside [writeXMLToStream\(\)](#) so that derived classes may add their own attributes to stream.*

- void [addFontProperties](#) ()

*Register all properties of this class.*

## Protected Attributes

- CodepointMap [d\\_cp\\_map](#)

*Contains mappings from code points to [Image](#) objects.*

- [String](#) [d\\_name](#)

*Name of this font.*

- [String](#) [d\\_fileName](#)

*Holds the name of the file used to create this font (either font file or imagset).*

- [String](#) [d\\_resourceGroup](#)

*Holds the name of the font file's resource group.*

- float [d\\_ascender](#)  
*maximal font ascender (pixels above the baseline)*
- float [d\\_descender](#)  
*maximal font descender (negative pixels below the baseline)*
- float [d\\_height](#)  
*(ascender - descender) + linegap*
- bool [d\\_autoScale](#)  
*true when auto-scaling is enabled.*
- float [d\\_horzScaling](#)  
*current horizontal scaling factor.*
- float [d\\_vertScaling](#)  
*current vertical scaling factor.*
- float [d\\_nativeHorzRes](#)  
*native horizontal resolution for this [Imageset](#).*
- float [d\\_nativeVertRes](#)  
*native vertical resolution for this [Imageset](#).*
- utf32 [d\\_maxCodepoint](#)  
*Maximal codepoint for font glyphs.*
- uint \* [d\\_glyphPageLoaded](#)  
*This bitmap holds information about loaded 'pages' of glyphs. A glyph page is a set of 256 codepoints, starting at 256-multiples. For example, the 1st glyph page is 0-255, fourth is 1024-1279 etc. When a specific glyph is required for painting, the corresponding bit is checked to see if the respective page has been rasterized. If not, the [rasterize\(\)](#) method is invoked, which prepares the glyphs from the respective glyph page for being painted.*

## Static Protected Attributes

- static [String d\\_defaultResourceGroup](#)  
*hold default resource group for font loading.*

## Friends

- class **FontManager**
- class **Font\_xmlHandler**
- class **FontProperties::NativeRes**
- class **FontProperties::Name**
- class **FontProperties::FileName**
- class **FontProperties::ResourceGroup**
- class **FontProperties::AutoScaled**

### 6.82.1 Detailed Description

Class that encapsulates text rendering functionality for a typeface.

A [Font](#) object is created for each unique typeface required. The [Font](#) class provides methods for loading typefaces from various sources, and then for outputting text via the [Renderer](#) object.

This class is not specific to any font renderer, it just provides the basic interfaces needed to manage fonts.

### 6.82.2 Constructor & Destructor Documentation

#### 6.82.2.1 CEGUI::Font::Font (const String & *name*, const String & *fontname*, const String & *resourceGroup* = "") [protected]

Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to set up all remaining font parameters after constructing the [Font](#) object, and finally calling the [load\(\)](#) method which will make font available for use. All font parameters that are not initialized are set to sensible default values.

##### Parameters:

- name* The unique name that will be used to identify this [Font](#).
- fontname* The filename of the font file, which contains the font data. This can be a TrueType, PostScript, bitmap font etc file.
- resourceGroup* Resource group identifier to be passed to the resource provider to load the font definition file.

#### 6.82.2.2 CEGUI::Font::Font (const XMLAttributes & *attributes*) [protected]

Constructs a new [Font](#) object and instantly loads it. The font is ready for use right after creation, there is no need to [load\(\)](#) it. All data required by this font is loaded from the provided [XMLAttributes](#) object.

##### Parameters:

- attributes* The XML attributes attached to this [Font](#).

##### Exceptions:

- [FileIOException](#) thrown if there was some problem accessing or parsing the file *filename*
- [InvalidRequestException](#) thrown if an invalid filename was provided.
- [AlreadyExistsException](#) thrown if a [Font Imageset](#) clashes with one already defined in the system.
- [GenericException](#) thrown if something goes wrong while accessing a true-type font referenced in file *filename*.
- [RendererException](#) thrown if the [Renderer](#) can't support a texture large enough to hold the requested glyph imagery.
- [MemoryException](#) thrown if allocation of imagery construction buffer fails.

### 6.82.3 Member Function Documentation

#### 6.82.3.1 const FontGlyph \* CEGUI::Font::getGlyphData (utf32 *codepoint*) [protected]

Return a pointer to the glyphDat struct for the given codepoint, or 0 if the codepoint does not have a glyph defined.

**Parameters:**

*codepoint* utf32 codepoint to return the glyphDat structure for.

**Returns:**

Pointer to the glyphDat struct for *codepoint*, or 0 if no glyph is defined for *codepoint*.

**6.82.3.2 void CEGUI::Font::rasterize (utf32 start\_codepoint, utf32 end\_codepoint)**  
[protected, virtual]

This function prepares a certain range of glyphs to be ready for displaying. This means that after returning from this function glyphs from `d_cp_map[start_codepoint]` to `d_cp_map[end_codepoint]` should have their `d_image` member set. If there is an error during rasterization of some glyph, it's okay to leave the `d_image` field set to NULL, in which case such glyphs will be skipped from display.

**Parameters:**

*start\_codepoint* The lowest codepoint that should be rasterized

*end\_codepoint* The highest codepoint that should be rasterized

Reimplemented in [CEGUI::FreeTypeFont](#).

**6.82.3.3 void CEGUI::Font::writeXMLToStream (XMLSerializer & xml\_stream) const**  
[protected]

Writes an xml representation of this [Font](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.82.3.4 virtual void CEGUI::Font::writeXMLToStream\_impl (XMLSerializer & xml\_stream) const**  
[protected, pure virtual]

Same as [writeXMLToStream\(\)](#) but called from inside [writeXMLToStream\(\)](#) so that derived classes may add their own attributes to stream.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

Implemented in [CEGUI::FreeTypeFont](#), and [CEGUI::PixmapFont](#).

### 6.82.3.5 `bool CEGUI::Font::isCodepointAvailable (utf32 cp) const [inline]`

Return whether this [Font](#) can draw the specified code-point.

#### Parameters:

*cp* utf32 code point that is the subject of the query.

#### Returns:

true if the font contains a mapping for code point *cp*, false if it does not contain a mapping for *cp*.

### 6.82.3.6 `size_t CEGUI::Font::drawText (const String & text, const Rect & draw_area, float z, const Rect & clip_rect, TextFormatting fmt, const ColourRect & colours, float x_scale = 1.0f, float y_scale = 1.0f)`

Draw text into a specified area of the display.

#### Parameters:

*text* [String](#) object containing the text to be drawn.

*draw\_area* [Rect](#) object describing the area of the display where the text is to be rendered. The text is not clipped to this [Rect](#), but is formatted using this [Rect](#) depending upon the option specified in *fmt*.

*z* flat value specifying the z co-ordinate for the drawn text.

*clip\_rect* [Rect](#) object describing the clipping area for the drawing. No drawing will occur outside this [Rect](#).

*fmt* One of the TextFormatting values specifying the text formatting required.

*colours* [ColourRect](#) object describing the colours to be applied when drawing the text. NB: The colours specified in here are applied to each glyph, rather than the text as a whole.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

#### Returns:

The number of lines output. NB: This does not consider clipping, so if all text was clipped, this would still return  $\geq 1$ .

### 6.82.3.7 `size_t CEGUI::Font::drawText (const String & text, const Rect & draw_area, float z, const Rect & clip_rect, TextFormatting fmt, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text into a specified area of the display using default colours.

#### Parameters:

*text* [String](#) object containing the text to be drawn.

*draw\_area* [Rect](#) object describing the area of the display where the text is to be rendered. The text is not clipped to this [Rect](#), but is formatted using this [Rect](#) depending upon the option specified in *fmt*.



*z* flat value specifying the z co-ordinate for the drawn text.

*clip\_rect* [Rect](#) object describing the clipping area for the drawing. No drawing will occur outside this [Rect](#).

*fmt* One of the TextFormatting values specifying the text formatting required.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

#### Returns:

The number of lines output. NB: This does not consider clipping, so if all text was clipped, this would still return >=1.

**6.82.3.8** `void CEGUI::Font::drawText (const String & text, const Rect & draw_area, float z, const Rect & clip_rect, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text into a specified area of the display with default colours and default formatting (LeftAligned).

#### Parameters:

*text* [String](#) object containing the text to be drawn.

*draw\_area* [Rect](#) object describing the area of the display where the text is to be rendered. The text is not clipped to this [Rect](#), but is formatted using this [Rect](#) depending upon the option specified in *fmt*.

*z* flat value specifying the z co-ordinate for the drawn text.

*clip\_rect* [Rect](#) object describing the clipping area for the drawing. No drawing will occur outside this [Rect](#).

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

#### Returns:

Nothing.

**6.82.3.9** `size_t CEGUI::Font::drawText (const String & text, const Rect & draw_area, float z, TextFormatting fmt, const ColourRect & colours, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text into a specified area of the display.

#### Parameters:

*text* [String](#) object containing the text to be drawn.

*draw\_area* [Rect](#) object describing the area of the display where the text is to be rendered. The text is formatted using this [Rect](#) depending upon the option specified in *fmt*. Additionally, the drawn text is clipped to be within this [Rect](#) (applies to non-word wrapped formatting where the text may otherwise have fallen outside this [Rect](#)).

*z* flat value specifying the z co-ordinate for the drawn text.

*fmt* One of the TextFormatting values specifying the text formatting required.

*colours* [ColourRect](#) object describing the colours to be applied when drawing the text. NB: The colours specified in here are applied to each glyph, rather than the text as a whole.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

#### Returns:

The number of lines output. NB: This does not consider clipping, so if all text was clipped, this would still return  $\geq 1$ .

#### 6.82.3.10 `size_t CEGUI::Font::drawText (const String & text, const Rect & draw_area, float z, TextFormatting fmt, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text into a specified area of the display with default colours.

#### Parameters:

*text* [String](#) object containing the text to be drawn.

*draw\_area* [Rect](#) object describing the area of the display where the text is to be rendered. The text is formatted using this [Rect](#) depending upon the option specified in *fmt*. Additionally, the drawn text is clipped to be within this [Rect](#) (applies to non-word wrapped formatting where the text may otherwise have fallen outside this [Rect](#)).

*z* flat value specifying the z co-ordinate for the drawn text.

*fmt* One of the TextFormatting values specifying the text formatting required.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

#### Returns:

The number of lines output. NB: This does not consider clipping, so if all text was clipped, this would still return  $\geq 1$ .

#### 6.82.3.11 `void CEGUI::Font::drawText (const String & text, const Rect & draw_area, float z, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text into a specified area of the display with default colours and default formatting (LeftAligned).

#### Parameters:

*text* [String](#) object containing the text to be drawn.

*draw\_area* [Rect](#) object describing the area of the display where the text is to be rendered. The text is formatted using this [Rect](#) depending upon the option specified in *fmt*. Additionally, the drawn text is clipped to be within this [Rect](#) (applies to non-word wrapped formatting where the text may otherwise have fallen outside this [Rect](#)).

*z* flat value specifying the z co-ordinate for the drawn text.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

#### Returns:

Nothing.

**6.82.3.12** `void CEGUI::Font::drawText (const String & text, const Vector3 & position, const Rect & clip_rect, const ColourRect & colours, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text at the specified location.

**Parameters:**

*text* [String](#) object containing the text to be drawn.

*position* [Vector3](#) object describing the location for the text. NB: The position specified here corresponds to the text baseline and not the top of any glyph. The baseline spacing required can be retrieved by calling [getBaseline\(\)](#).

*clip\_rect* [Rect](#) object describing the clipping area for the drawing. No drawing will occur outside this [Rect](#).

*colours* [ColourRect](#) object describing the colours to be applied when drawing the text. NB: The colours specified in here are applied to each glyph, rather than the text as a whole.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

**Returns:**

Nothing.

**6.82.3.13** `void CEGUI::Font::drawText (const String & text, const Vector3 & position, const Rect & clip_rect, float x_scale = 1.0f, float y_scale = 1.0f) [inline]`

Draw text at the specified location with default colours.

**Parameters:**

*text* [String](#) object containing the text to be drawn.

*position* [Vector3](#) object describing the location for the text. NB: The position specified here corresponds to the text baseline and not the top of any glyph. The baseline spacing required can be retrieved by calling [getBaseline\(\)](#).

*clip\_rect* [Rect](#) object describing the clipping area for the drawing. No drawing will occur outside this [Rect](#).

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

*y\_scale* Scaling factor to be applied to each glyph's y axis, where 1.0f is considered to be 'normal'.

**Returns:**

Nothing.

**6.82.3.14** `void CEGUI::Font::setNativeResolution (const Size & size) [virtual]`

Set the native resolution for this [Font](#).

**Parameters:**

*size* [Size](#) object describing the new native screen resolution for this [Font](#).

**Returns:**

Nothing

**6.82.3.15 void CEGUI::Font::notifyScreenResolution (const Size & size) [virtual]**

Notify the [Font](#) of the current (usually new) display resolution.

**Parameters:**

*size* [Size](#) object describing the display resolution

**Returns:**

Nothing

**6.82.3.16 float CEGUI::Font::getLineSpacing (float y\_scale = 1.0f) const [inline]**

Return the pixel line spacing value for.

**Parameters:**

*y\_scale* Scaling factor to be applied to the line spacing, where 1.0f is considered to be 'normal'.

**Returns:**

Number of pixels between vertical base lines, i.e. The minimum pixel space between two lines of text.

**6.82.3.17 float CEGUI::Font::getFontHeight (float y\_scale = 1.0f) const [inline]**

return the exact pixel height of the font.

**Parameters:**

*y\_scale* Scaling factor to be applied to the height, where 1.0f is considered to be 'normal'.

**Returns:**

float value describing the pixel height of the font without any additional padding.

**6.82.3.18 float CEGUI::Font::getBaseline (float y\_scale = 1.0f) const [inline]**

Return the number of pixels from the top of the highest glyph to the baseline.

**Parameters:**

*y\_scale* Scaling factor to be applied to the baseline distance, where 1.0f is considered to be 'normal'.

**Returns:**

pixel spacing from top of front glyphs to baseline

**6.82.3.19 float CEGUI::Font::getTextExtent (const String & *text*, float *x\_scale* = 1.0f)**

Return the pixel width of the specified text if rendered with this [Font](#).

**Parameters:**

*text* [String](#) object containing the text to return the rendered pixel width for.

*x\_scale* Scaling factor to be applied to each glyph's x axis when measuring the extent, where 1.0f is considered to be 'normal'.

**Returns:**

Number of pixels that *text* will occupy when rendered with this [Font](#).

**6.82.3.20 size\_t CEGUI::Font::getCharAtPixel (const String & *text*, float *pixel*, float *x\_scale* = 1.0f) [inline]**

Return the index of the closest text character in [String](#) *text* that corresponds to pixel location *pixel* if the text were rendered.

**Parameters:**

*text* [String](#) object containing the text.

*pixel* Specifies the (horizontal) pixel offset to return the character index for.

*x\_scale* Scaling factor to be applied to each glyph's x axis when measuring the text extent, where 1.0f is considered to be 'normal'.

**Returns:**

Returns a character index into [String](#) *text* for the character that would be rendered closest to horizontal pixel offset *pixel* if the text were to be rendered via this [Font](#). Range of the return is from 0 to text.length(), so may actually return an index past the end of the string, which indicates *pixel* was beyond the last character.

**6.82.3.21 size\_t CEGUI::Font::getCharAtPixel (const String & *text*, size\_t *start\_char*, float *pixel*, float *x\_scale* = 1.0f)**

Return the index of the closest text character in [String](#) *text*, starting at character index *start\_char*, that corresponds to pixel location *pixel* if the text were to be rendered.

**Parameters:**

*text* [String](#) object containing the text.

*start\_char* index of the first character to consider. This is the lowest value that will be returned from the call.

*pixel* Specifies the (horizontal) pixel offset to return the character index for.

*x\_scale* Scaling factor to be applied to each glyph's x axis when measuring the text extent, where 1.0f is considered to be 'normal'.

**Returns:**

Returns a character index into [String](#) *text* for the character that would be rendered closest to horizontal pixel offset *pixel* if the text were to be rendered via this [Font](#). Range of the return is from 0 to text.length(), so may actually return an index past the end of the string, which indicates *pixel* was beyond the last character.

**6.82.3.22** `size_t CEGUI::Font::getFormattedLineCount (const String & text, const Rect & format_area, TextFormatting fmt, float x_scale = 1.0f)`

Return the number of lines the given text would be formatted to.

Since text formatting can result in multiple lines of text being output, it can be useful to know how many lines would be output without actually rendering the text.

**Parameters:**

*text* [String](#) object containing the text to be measured.

*format\_area* [Rect](#) object describing the area to be used when formatting the text depending upon the option specified in *fmt*.

*fmt* One of the TextFormatting values specifying the text formatting required.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

**Returns:**

The number of lines produced from the specified formatting

**6.82.3.23** `float CEGUI::Font::getFormattedTextExtent (const String & text, const Rect & format_area, TextFormatting fmt, float x_scale = 1.0f)`

Return the horizontal pixel extent given text would be formatted to.

The value return by this method is basically the extent of the widest line within the formatted text.

**Parameters:**

*text* [String](#) object containing the text to be measured.

*format\_area* [Rect](#) object describing the area to be used when formatting the text depending upon the option specified in *fmt*.

*fmt* One of the TextFormatting values specifying the text formatting required.

*x\_scale* Scaling factor to be applied to each glyph's x axis, where 1.0f is considered to be 'normal'.

**Returns:**

The widest pixel extent of the lines produced from the specified formatting.

**6.82.3.24** `static void CEGUI::Font::setDefaultResourceGroup (const String & resourceGroup)`  
`[inline, static]`

Sets the default resource group to be used when loading font data.

**Parameters:**

*resourceGroup* [String](#) describing the default resource group identifier to be used.

**Returns:**

Nothing.

**6.82.3.25** `static const String& CEGUI::Font::getDefaultResourceGroup (void)` `[inline, static]`

Returns the default resource group currently set for Fonts.

**Returns:**

[String](#) describing the default resource group identifier that will be used when loading font data.

## 6.82.4 Member Data Documentation

**6.82.4.1** `uint* CEGUI::Font::d_glyphPageLoaded` `[protected]`

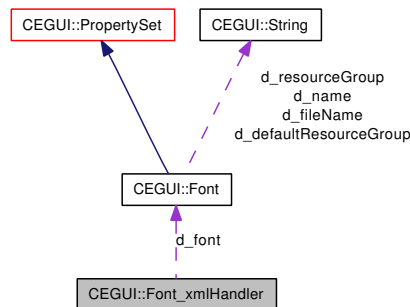
This bitmap holds information about loaded 'pages' of glyphs. A glyph page is a set of 256 codepoints, starting at 256-multiples. For example, the 1st glyph page is 0-255, fourth is 1024-1279 etc. When a specific glyph is required for painting, the corresponding bit is checked to see if the respective page has been rasterized. If not, the [rasterize\(\)](#) method is invoked, which prepares the glyphs from the respective glyph page for being painted.

This array is big enough to hold at least max\_codepoint bits. If this member is NULL, all glyphs are considered pre-rasterized.

## 6.83 CEGUI::Font\_xmlHandler Class Reference

Handler class used to parse the [Font](#) XML files using SAX2.

Collaboration diagram for CEGUI::Font\_xmlHandler:



### Public Member Functions

- [Font\\_xmlHandler](#) ()  
*Constructor for Font::xmlHandler objects.*
- virtual [~Font\\_xmlHandler](#) (void)  
*Destructor for Font::xmlHandler objects.*
- virtual void [elementStart](#) (const [String](#) &element, const [XMLAttributes](#) &attributes)  
*document processing (only care about elements, schema validates format)*
- virtual void [elementEnd](#) (const [String](#) &element)

### Public Attributes

- [Font](#) \* [d\\_font](#)  
*Font object that we are helping to build.*

#### 6.83.1 Detailed Description

Handler class used to parse the [Font](#) XML files using SAX2.

#### 6.83.2 Constructor & Destructor Documentation

##### 6.83.2.1 CEGUI::Font\_xmlHandler::Font\_xmlHandler () [inline]

Constructor for Font::xmlHandler objects.

##### Parameters:

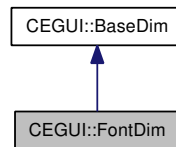
*font* Pointer to the [Font](#) object creating this xmlHandler object



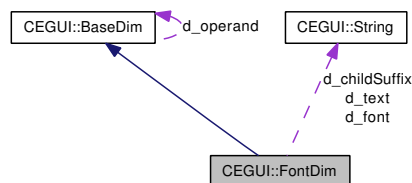
## 6.84 CEGUI::FontDim Class Reference

**Dimension** type that represents some metric of a **Font**. Implements **BaseDim** interface.

Inheritance diagram for CEGUI::FontDim:



Collaboration diagram for CEGUI::FontDim:



### Public Member Functions

- **FontDim** (const **String** &name, const **String** &font, const **String** &text, **FontMetricType** metric, float padding=0)

*Constructor.*

### Protected Member Functions

- float **getValue\_impl** (const **Window** &wnd) const  
*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically.*
- float **getValue\_impl** (const **Window** &wnd, const **Rect** &container) const  
*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically by **BaseDim**.*
- void **writeXMLElementName\_impl** (**XMLSerializer** &xml\_stream) const  
*Implementataion method to output real xml element name.*
- void **writeXMLElementAttributes\_impl** (**XMLSerializer** &xml\_stream) const  
*Implementataion method to create the element attributes.*
- **BaseDim** \* **clone\_impl** () const  
*Implementataion method to return a clone of this sub-class of **BaseDim**. This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by **BaseDim**.*

### 6.84.1 Detailed Description

[Dimension](#) type that represents some metric of a [Font](#). Implements [BaseDim](#) interface.

### 6.84.2 Constructor & Destructor Documentation

#### 6.84.2.1 CEGUI::FontDim::FontDim (const String & *name*, const String & *font*, const String & *text*, FontMetricType *metric*, float *padding* = 0)

Constructor.

##### Parameters:

*name* [String](#) holding the name suffix of the window to be accessed to obtain the font and / or text strings to be used when these items are not explicitly given.

*font* [String](#) holding the name of the font to use for this dimension. If the string is empty, the font assigned to the window passed to `getValue` will be used.

*text* [String](#) holding the text to be measured for horizontal extent. If this is empty, the text from the window passed to `getValue` will be used.

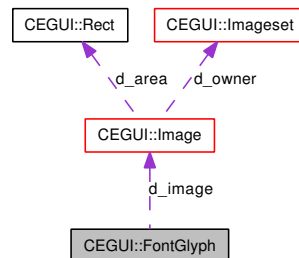
*metric* One of the `FontMetricType` values indicating what we should represent.

*padding* constant pixel padding value to be added.

## 6.85 CEGUI::FontGlyph Class Reference

internal class representing a single font glyph.

Collaboration diagram for CEGUI::FontGlyph:



### Public Member Functions

- [FontGlyph \(\)](#)  
*construct an empty uninitialized [FontGlyph](#)*
- [FontGlyph \(float advance\)](#)  
*[FontGlyph](#) constructor.*
- [FontGlyph \(float advance, const \[Image\]\(#\) \\*image\)](#)  
*A better :-> [FontGlyph](#) constructor.*
- const [Image](#) \* [getImage \(\)](#) const  
*Return the [CEGUI::Image](#) object rendered for this glyph.*
- const [ImageSet](#) \* [getImageSet \(\)](#) const  
*Return the parent [CEGUI::ImageSet](#) object for this glyph.*
- [Size](#) [getSize \(float x\\_scale, float y\\_scale\)](#) const  
*Return the scaled pixel size of the glyph.*
- float [getWidth \(float x\\_scale\)](#) const  
*Return the scaled width of the glyph.*
- float [getHeight \(float y\\_scale\)](#) const  
*Return the scaled height of the glyph.*
- float [getRenderedAdvance \(float x\\_scale\)](#) const  
*Return the rendered advance value for this glyph.*
- float [getAdvance \(float x\\_scale=1.0\)](#) const  
*Return the horizontal advance value for the glyph.*
- void [setAdvance \(float advance\)](#)  
*Set the horizontal advance value for the glyph.*

- void `setImage` (const `Image` \*image)

*Set the `CEGUI::Image` object rendered for this glyph.*

### 6.85.1 Detailed Description

internal class representing a single font glyph.

For TrueType fonts initially all `FontGlyph`'s are empty (`getImage()` will return `NULL`), but they are filled by demand.

### 6.85.2 Member Function Documentation

#### 6.85.2.1 `float CEGUI::FontGlyph::getRenderedAdvance (float x_scale) const` `[inline]`

Return the rendered advance value for this glyph.

The rendered advance value is the total number of pixels from the current pen position that will be occupied by this glyph when rendered.

#### 6.85.2.2 `float CEGUI::FontGlyph::getAdvance (float x_scale = 1.0) const` `[inline]`

Return the horizontal advance value for the glyph.

The returned value is the number of pixels the pen should move horizontally to position itself ready to render the next glyph. This is not always the same as the glyph image width or rendered advance, since it allows for horizontal overhangs.

## 6.86 CEGUI::FontManager Class Reference

Class providing a shared library of [Font](#) objects to the system.

### Public Types

- typedef [ConstBaseIterator](#)< FontRegistry > [FontIterator](#)

### Public Member Functions

- [FontManager](#) (void)  
*Constructor for [FontManager](#) objects.*
- [~FontManager](#) (void)  
*Destructor for [FontManager](#) objects.*
- [Font](#) \* [createFont](#) (const [String](#) &filename, const [String](#) &resourceGroup="")  
*Creates a new font from a font definition file, and returns a pointer to the new [Font](#) object.*
- [Font](#) \* [createFont](#) (const [String](#) &type, const [String](#) &name, const [String](#) &fontname, const [String](#) &resourceGroup="")  
*Creates a new [Font](#) based on a true-type font, and returns a pointer to the new [Font](#) object.*
- [Font](#) \* [createFont](#) (const [String](#) &type, const [XMLAttributes](#) &attributes)  
*Create a new font object given its type and the XML attributes.*
- void [destroyFont](#) (const [String](#) &name)  
*Destroy's the font with the given name.*
- void [destroyFont](#) ([Font](#) \*font)  
*Destroys the given [Font](#) object.*
- void [destroyAllFonts](#) (void)  
*Destroys all [Font](#) objects registered in the system.*
- bool [isFontPresent](#) (const [String](#) &name) const  
*Checks the existence of a given font.*
- [Font](#) \* [getFont](#) (const [String](#) &name) const  
*Returns a pointer to the font object with the specified name.*
- void [notifyScreenResolution](#) (const [Size](#) &size)  
*Notify the [FontManager](#) of the current (usually new) display resolution.*
- void [writeFontToStream](#) (const [String](#) &name, [OutStream](#) &out\_stream) const  
*Writes a full XML font file for the specified [Font](#) to the given [OutStream](#).*
- [FontIterator](#) [getIterator](#) (void) const  
*Return a [FontManager::FontIterator](#) object to iterate over the available [Font](#) objects.*

### 6.86.1 Detailed Description

Class providing a shared library of [Font](#) objects to the system.

The [FontManager](#) is used to create, access, and destroy [Font](#) objects. The idea is that the [FontManager](#) will function as a central repository for [Font](#) objects used within the GUI system, and that those [Font](#) objects can be accessed, via a unique name, by any interested party within the system.

### 6.86.2 Member Function Documentation

#### 6.86.2.1 [Font](#) \* CEGUI::FontManager::createFont (const [String](#) & *filename*, const [String](#) & *resourceGroup* = "")

Creates a new font from a font definition file, and returns a pointer to the new [Font](#) object.

##### Parameters:

*filename* [String](#) object containing the filename of a 'font definition file' what will be used to create the new font

*resourceGroup* Resource group identifier to pass to the resource provider when loading the font definition file.

##### Returns:

Pointer the the newly created [Font](#) object

##### Exceptions:

[FileIOException](#) thrown if there was some problem accessing or parsing the file *filename*

[InvalidRequestException](#) thrown if an invalid filename was provided.

[AlreadyExistsException](#) thrown if a [Font](#) already exists with the name specified, or if a font [Imageset](#) clashes with one already defined in the system.

[GenericException](#) thrown if something goes wrong while accessing a true-type font referenced in file *filename*.

[RendererException](#) thrown if the [Renderer](#) can't support a texture large enough to hold the requested glyph imagery.

[MemoryException](#) thrown if allocation of imagery construction buffer fails.

#### 6.86.2.2 [Font](#) \* CEGUI::FontManager::createFont (const [String](#) & *type*, const [String](#) & *name*, const [String](#) & *fontname*, const [String](#) & *resourceGroup* = "")

Creates a new [Font](#) based on a true-type font, and returns a pointer to the new [Font](#) object.

##### Parameters:

*type* [String](#) object containing the type of the font to be created (same as in the "Type" attribute of the font XML).

*name* [String](#) object containing a unique name for the new font.

*fontname* [String](#) object containing the name and path of the true-type font to access.

*resourceGroup* Resource group identifier to be passed to the resource provider when loading the font definition file.

**Returns:**

Pointer to the newly created [Font](#) object.

**Exceptions:**

[AlreadyExistsException](#) thrown if a [Font](#) already exists with the name specified, or if a font [Imageset](#) clashes with one already defined in the system.

[GenericException](#) thrown if something goes wrong while accessing a true-type font referenced in file *fontname*.

[RendererException](#) thrown if the [Renderer](#) can't support a texture large enough to hold the requested glyph imagery.

[MemoryException](#) thrown if allocation of imagery construction buffer fails.

**6.86.2.3 Font \* CEGUI::FontManager::createFont (const String & type, const XMLAttributes & attributes)**

Create a new font object given its type and the XML attributes.

**Returns:**

The new font object or NULL.

**6.86.2.4 void CEGUI::FontManager::destroyFont (const String & name)**

Destroy's the font with the given name.

**Parameters:**

*name* [String](#) object containing the name of the font to be destroyed. If the specified font does not exist, nothing happens.

**Returns:**

Nothing

**6.86.2.5 void CEGUI::FontManager::destroyFont (Font \* font)**

Destroys the given [Font](#) object.

**Parameters:**

*font* Pointer to the [Font](#) to be destroyed. If no such [Font](#) exists, nothing happens.

**Returns:**

Nothing.

**6.86.2.6 void CEGUI::FontManager::destroyAllFonts (void)**

Destroys all [Font](#) objects registered in the system.

**Returns:**

Nothing

**6.86.2.7 bool CEGUI::FontManager::isFontPresent (const String & name) const**

Checks the existence of a given font.

**Parameters:**

*name* [String](#) object holding the name of the [Font](#) object to look for.

**Returns:**

true if a [Font](#) object named *name* exists in the system, false if no such font exists.

**6.86.2.8 Font \* CEGUI::FontManager::getFont (const String & name) const**

Returns a pointer to the font object with the specified name.

**Parameters:**

*name* [String](#) object containing the name of the [Font](#) object to be returned

**Returns:**

Pointer to the requested [Font](#) object

**Exceptions:**

[UnknownObjectException](#) Thrown if no font with the given name exists.

**6.86.2.9 void CEGUI::FontManager::notifyScreenResolution (const Size & size)**

Notify the [FontManager](#) of the current (usually new) display resolution.

**Parameters:**

*size* [Size](#) object describing the display resolution

**Returns:**

Nothing



**6.86.2.10 void CEGUI::FontManager::writeFontToStream (const String & *name*, OutStream & *out\_stream*) const**

Writes a full XML font file for the specified [Font](#) to the given OutStream.

**Parameters:**

*name* [String](#) holding the name of the [Font](#) to be written to the stream.

*out\_stream* OutStream (std::ostream based) object where data is to be sent.

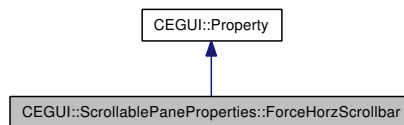
**Returns:**

Nothing.

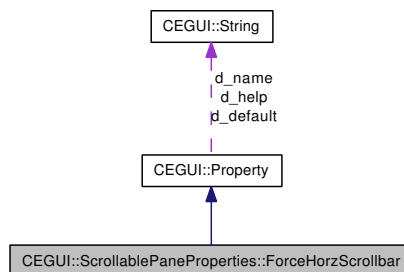
## 6.87 CEGUI::ScrollablePaneProperties::ForceHorzScrollbar Class Reference

[Property](#) to access the setting which controls whether the horizontal scroll bar will always be displayed, or only displayed when it is required.

Inheritance diagram for CEGUI::ScrollablePaneProperties::ForceHorzScrollbar:



Collaboration diagram for CEGUI::ScrollablePaneProperties::ForceHorzScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.87.1 Detailed Description

[Property](#) to access the setting which controls whether the horizontal scroll bar will always be displayed, or only displayed when it is required.

#### Usage:

- Name: [ForceHorzScrollbar](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate that the horizontal scroll bar will always be shown.
- "False" to indicate that the horizontal scroll bar will only be shown when it is needed.

## 6.87.2 Member Function Documentation

### 6.87.2.1 String CEGUI::ScrollablePaneProperties::ForceHorzScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.87.2.2 void CEGUI::ScrollablePaneProperties::ForceHorzScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

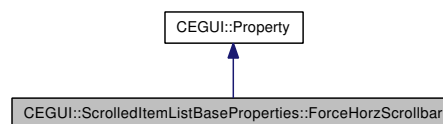
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

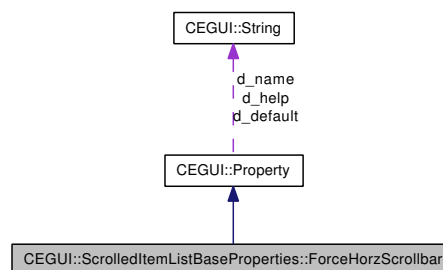
## 6.88 CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar Class Reference

[Property](#) to access the state of the force horizontal scrollbar setting.

Inheritance diagram for CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar:



Collaboration diagram for CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.88.1 Detailed Description

[Property](#) to access the state of the force horizontal scrollbar setting.

#### Usage:

- Name: [ForceHorzScrollbar](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate that the horizontal scrollbar should always be shown.
- "False" to indicate that the horizontal scrollbar should only be shown when needed.

## 6.88.2 Member Function Documentation

### 6.88.2.1 String CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.88.2.2 void CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

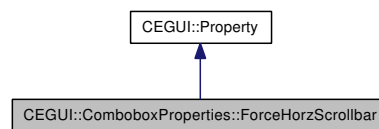
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

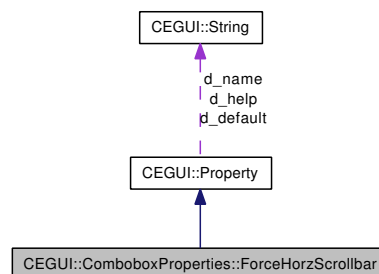
## 6.89 CEGUI::ComboboxProperties::ForceHorzScrollbar Class Reference

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

Inheritance diagram for CEGUI::ComboboxProperties::ForceHorzScrollbar:



Collaboration diagram for CEGUI::ComboboxProperties::ForceHorzScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.89.1 Detailed Description

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

**Usage:**

- Name: [ForceHorzScrollbar](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that the horizontal scroll bar will always be shown.
- "False" to indicate that the horizontal scroll bar will only be shown when it is needed.

## 6.89.2 Member Function Documentation

### 6.89.2.1 String CEGUI::ComboboxProperties::ForceHorzScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.89.2.2 void CEGUI::ComboboxProperties::ForceHorzScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

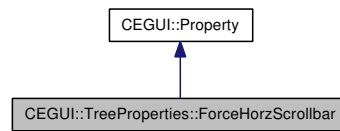
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

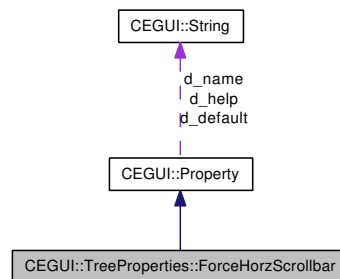
## 6.90 CEGUI::TreeProperties::ForceHorzScrollbar Class Reference

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

Inheritance diagram for CEGUI::TreeProperties::ForceHorzScrollbar:



Collaboration diagram for CEGUI::TreeProperties::ForceHorzScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.90.1 Detailed Description

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

Usage:

- Name: [ForceHorzScrollbar](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate that the horizontal scroll bar will always be shown.
- "False" to indicate that the horizontal scroll bar will only be shown when it is needed.



## 6.90.2 Member Function Documentation

### 6.90.2.1 String CEGUI::TreeProperties::ForceHorzScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.90.2.2 void CEGUI::TreeProperties::ForceHorzScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

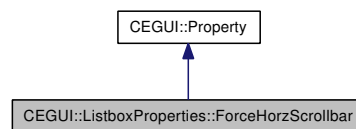
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

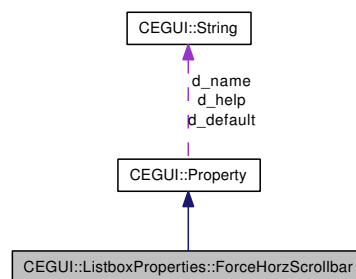
## 6.91 CEGUI::ListboxProperties::ForceHorzScrollbar Class Reference

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

Inheritance diagram for CEGUI::ListboxProperties::ForceHorzScrollbar:



Collaboration diagram for CEGUI::ListboxProperties::ForceHorzScrollbar:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.91.1 Detailed Description

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

Usage:

- Name: [ForceHorzScrollbar](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate that the horizontal scroll bar will always be shown.
- "False" to indicate that the horizontal scroll bar will only be shown when it is needed.

## 6.91.2 Member Function Documentation

### 6.91.2.1 String CEGUI::ListboxProperties::ForceHorzScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.91.2.2 void CEGUI::ListboxProperties::ForceHorzScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

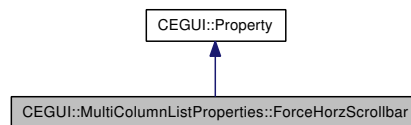
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

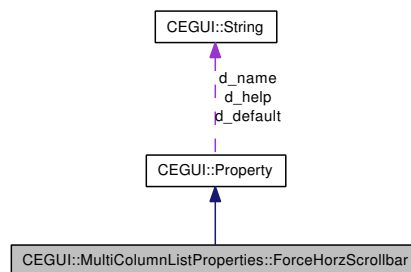
## 6.92 CEGUI::MultiColumnListProperties::ForceHorzScrollbar Class Reference

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

Inheritance diagram for CEGUI::MultiColumnListProperties::ForceHorzScrollbar:



Collaboration diagram for CEGUI::MultiColumnListProperties::ForceHorzScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.92.1 Detailed Description

[Property](#) to access the 'always show' setting for the horizontal scroll bar of the list box.

#### Usage:

- Name: [ForceHorzScrollbar](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate that the horizontal scroll bar will always be shown.
- "False" to indicate that the horizontal scroll bar will only be shown when it is needed.

## 6.92.2 Member Function Documentation

### 6.92.2.1 String CEGUI::MultiColumnListProperties::ForceHorzScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.92.2.2 void CEGUI::MultiColumnListProperties::ForceHorzScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

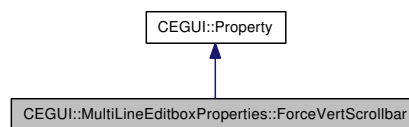
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

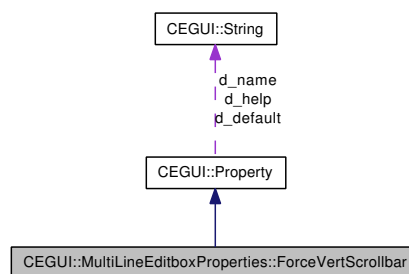
## 6.93 CEGUI::MultiLineEditboxProperties::ForceVertScrollbar Class Reference

[Property](#) to access the 'always show' setting for the vertical scroll bar of the box.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.93.1 Detailed Description

[Property](#) to access the 'always show' setting for the vertical scroll bar of the box.

#### Usage:

- Name: [ForceVertScrollbar](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate that the vertical scroll bar will always be shown.
- "False" to indicate that the vertical scroll bar will only be shown when it is needed.

## 6.93.2 Member Function Documentation

### 6.93.2.1 String CEGUI::MultiLineEditboxProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.93.2.2 void CEGUI::MultiLineEditboxProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

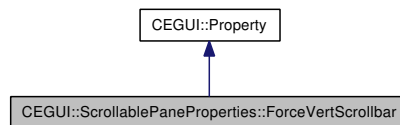
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

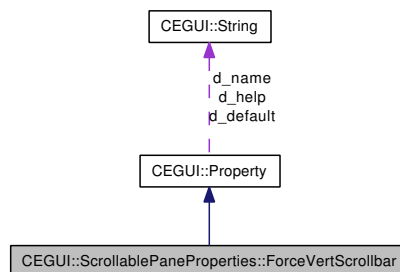
## 6.94 CEGUI::ScrollablePaneProperties::ForceVertScrollbar Class Reference

[Property](#) to access the setting which controls whether the vertical scroll bar will always be displayed, or only displayed when it is required.

Inheritance diagram for CEGUI::ScrollablePaneProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::ScrollablePaneProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.94.1 Detailed Description

[Property](#) to access the setting which controls whether the vertical scroll bar will always be displayed, or only displayed when it is required.

##### Usage:

- Name: [ForceVertScrollbar](#)
- Format: "[text]"

##### Where [Text] is:

- "True" to indicate that the vertical scroll bar will always be shown.
- "False" to indicate that the vertical scroll bar will only be shown when it is needed.



## 6.94.2 Member Function Documentation

### 6.94.2.1 String CEGUI::ScrollablePaneProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.94.2.2 void CEGUI::ScrollablePaneProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

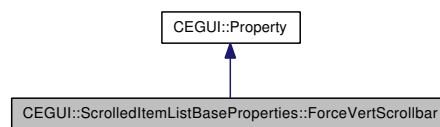
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

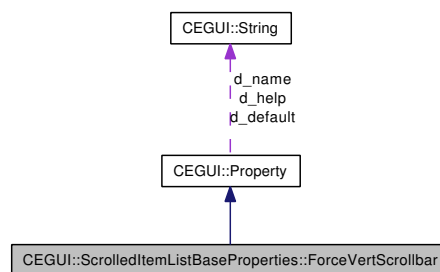
## 6.95 CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar Class Reference

[Property](#) to access the state of the force vertical scrollbar setting.

Inheritance diagram for CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.95.1 Detailed Description

[Property](#) to access the state of the force vertical scrollbar setting.

#### Usage:

- Name: [ForceVertScrollbar](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate that the vertical scrollbar should always be shown.
- "False" to indicate that the vertical scrollbar should only be shown when needed.

## 6.95.2 Member Function Documentation

### 6.95.2.1 String CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.95.2.2 void CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

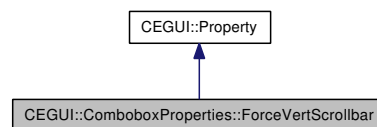
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

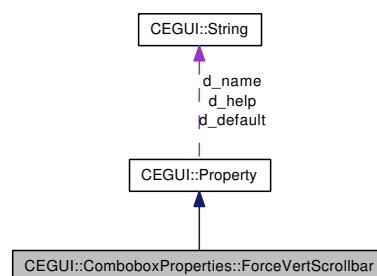
## 6.96 CEGUI::ComboboxProperties::ForceVertScrollbar Class Reference

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

Inheritance diagram for CEGUI::ComboboxProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::ComboboxProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.96.1 Detailed Description

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

**Usage:**

- Name: [ForceVertScrollbar](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that the vertical scroll bar will always be shown.
- "False" to indicate that the vertical scroll bar will only be shown when it is needed.

## 6.96.2 Member Function Documentation

### 6.96.2.1 String CEGUI::ComboboxProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.96.2.2 void CEGUI::ComboboxProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

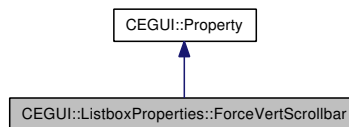
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

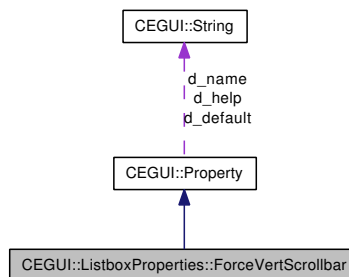
## 6.97 CEGUI::ListboxProperties::ForceVertScrollbar Class Reference

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

Inheritance diagram for CEGUI::ListboxProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::ListboxProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.97.1 Detailed Description

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

**Usage:**

- Name: [ForceVertScrollbar](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that the vertical scroll bar will always be shown.
- "False" to indicate that the vertical scroll bar will only be shown when it is needed.

## 6.97.2 Member Function Documentation

### 6.97.2.1 String CEGUI::ListboxProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.97.2.2 void CEGUI::ListboxProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

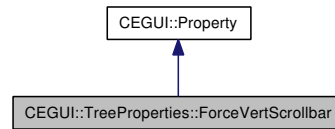
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

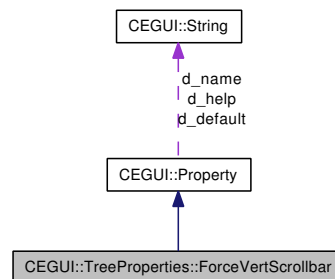
## 6.98 CEGUI::TreeProperties::ForceVertScrollbar Class Reference

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

Inheritance diagram for CEGUI::TreeProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::TreeProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.98.1 Detailed Description

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

Usage:

- Name: [ForceVertScrollbar](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate that the vertical scroll bar will always be shown.
- "False" to indicate that the vertical scroll bar will only be shown when it is needed.



## 6.98.2 Member Function Documentation

### 6.98.2.1 String CEGUI::TreeProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.98.2.2 void CEGUI::TreeProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

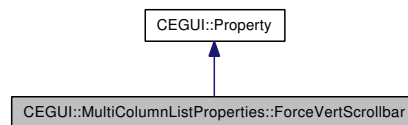
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

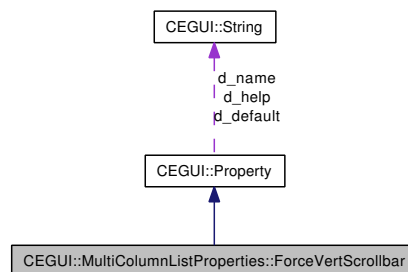
## 6.99 CEGUI::MultiColumnListProperties::ForceVertScrollbar Class Reference

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

Inheritance diagram for CEGUI::MultiColumnListProperties::ForceVertScrollbar:



Collaboration diagram for CEGUI::MultiColumnListProperties::ForceVertScrollbar:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.99.1 Detailed Description

[Property](#) to access the 'always show' setting for the vertical scroll bar of the list box.

**Usage:**

- Name: [ForceVertScrollbar](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that the vertical scroll bar will always be shown.
- "False" to indicate that the vertical scroll bar will only be shown when it is needed.

## 6.99.2 Member Function Documentation

### 6.99.2.1 String CEGUI::MultiColumnListProperties::ForceVertScrollbar::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.99.2.2 void CEGUI::MultiColumnListProperties::ForceVertScrollbar::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

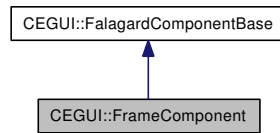
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

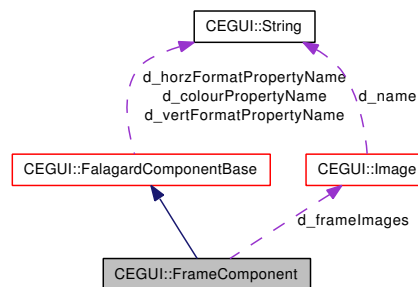
## 6.100 CEGUI::FrameComponent Class Reference

Class that encapsulates information for a frame with background (9 images in total).

Inheritance diagram for CEGUI::FrameComponent:



Collaboration diagram for CEGUI::FrameComponent:



### Public Member Functions

- [FrameComponent](#) ()  
*Constructor.*
- [VerticalFormatting](#) [getBackgroundVerticalFormatting](#) () const  
*Return the current vertical formatting setting for this [FrameComponent](#).*
- void [setBackgroundVerticalFormatting](#) ([VerticalFormatting](#) fmt)  
*Set the vertical formatting setting for this [FrameComponent](#).*
- [HorizontalFormatting](#) [getBackgroundHorizontalFormatting](#) () const  
*Return the current horizontal formatting setting for this [FrameComponent](#).*
- void [setBackgroundHorizontalFormatting](#) ([HorizontalFormatting](#) fmt)  
*Set the horizontal formatting setting for this [FrameComponent](#).*
- const [Image](#) \* [getImage](#) ([FrameImageComponent](#) part) const  
*Return the [Image](#) object that will be drawn by this [FrameComponent](#) for a specified frame part.*
- void [setImage](#) ([FrameImageComponent](#) part, const [Image](#) \*image)  
*Set the [Image](#) that will be drawn by this [ImageryComponent](#).*
- void [setImage](#) ([FrameImageComponent](#) part, const [String](#) &imageset, const [String](#) &image)  
*Set the [Image](#) that will be drawn by this [FrameComponent](#).*

- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const

*Writes an xml representation of this [FrameComponent](#) to out\_stream.*

## Protected Member Functions

- void [render\\_impl](#) ([Window](#) &srcWindow, [Rect](#) &destRect, float base\_z, const [CEGUI::ColourRect](#) \*modColours, const [Rect](#) \*clipper, bool clipToDisplay) const

*Method to do main render caching work.*

- void [doBackgroundRender](#) ([Window](#) &srcWindow, [Rect](#) &destRect, float base\_z, const [ColourRect](#) &colours, const [Rect](#) \*clipper, bool clipToDisplay) const

## Protected Attributes

- [VerticalFormatting](#) [d\\_vertFormatting](#)

*Vertical formatting to be applied when rendering the background for the component.*

- [HorizontalFormatting](#) [d\\_horzFormatting](#)

*Horizontal formatting to be applied when rendering the background for the component.*

- const [Image](#) \* [d\\_frameImages](#) [[FIC\\_FRAME\\_IMAGE\\_COUNT](#)]

*Array that holds the assigned images.*

### 6.100.1 Detailed Description

Class that encapsulates information for a frame with background (9 images in total).

Corner images are always drawn at their natural size, edges are stretched between the corner pieces for a particular edge, the background image will cover the inner rectangle formed by the edge images and can be stretched or tiled in either dimension.

### 6.100.2 Member Function Documentation

#### 6.100.2.1 [VerticalFormatting](#) [CEGUI::FrameComponent::getBackgroundVerticalFormatting](#) () const

Return the current vertical formatting setting for this [FrameComponent](#).

#### Returns:

One of the [VerticalFormatting](#) enumerated values.

**6.100.2.2 void CEGUI::FrameComponent::setBackgroundVerticalFormatting (VerticalFormatting *fmt*)**

Set the vertical formatting setting for this [FrameComponent](#).

**Parameters:**

*fmt* One of the VerticalFormatting enumerated values.

**Returns:**

Nothing.

**6.100.2.3 HorizontalFormatting  
CEGUI::FrameComponent::getBackgroundHorizontalFormatting ()  
const**

Return the current horizontal formatting setting for this [FrameComponent](#).

**Returns:**

One of the HorizontalFormatting enumerated values.

**6.100.2.4 void CEGUI::FrameComponent::setBackgroundHorizontalFormatting (HorizontalFormatting *fmt*)**

Set the horizontal formatting setting for this [FrameComponent](#).

**Parameters:**

*fmt* One of the HorizontalFormatting enumerated values.

**Returns:**

Nothing.

**6.100.2.5 const Image \* CEGUI::FrameComponent::getImage (FrameImageComponent *part*)  
const**

Return the [Image](#) object that will be drawn by this [FrameComponent](#) for a specified frame part.

**Parameters:**

*part* One of the FrameImageComponent enumerated values specifying the component image to be accessed.

**Returns:**

[Image](#) object.

**6.100.2.6 void CEGUI::FrameComponent::setImage (FrameImageComponent *part*, const Image \* *image*)**

Set the [Image](#) that will be drawn by this [ImageryComponent](#).

**Parameters:**

*part* One of the FrameImageComponent enumerated values specifying the component image to be accessed.

*image* Pointer to the [Image](#) object to be drawn by this [FrameComponent](#).

**Returns:**

Nothing.

**6.100.2.7 void CEGUI::FrameComponent::setImage (FrameImageComponent *part*, const String & *imageset*, const String & *image*)**

Set the [Image](#) that will be drawn by this [FrameComponent](#).

**Parameters:**

*part* One of the FrameImageComponent enumerated values specifying the component image to be accessed.

*imageset* [String](#) holding the name of the Imagset that contains the [Image](#) to be rendered.

*image* [String](#) holding the name of the [Image](#) to be rendered.

**Returns:**

Nothing.

**6.100.2.8 void CEGUI::FrameComponent::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [FrameComponent](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

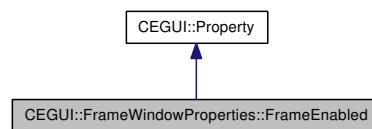
**Returns:**

Nothing.

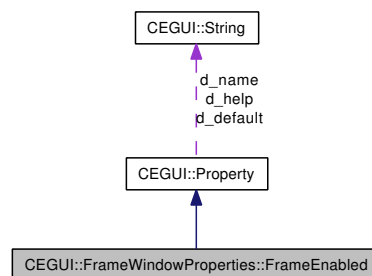
## 6.101 CEGUI::FrameWindowProperties::FrameEnabled Class Reference

[Property](#) to access the setting for whether the window frame will be displayed.

Inheritance diagram for CEGUI::FrameWindowProperties::FrameEnabled:



Collaboration diagram for CEGUI::FrameWindowProperties::FrameEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.101.1 Detailed Description

[Property](#) to access the setting for whether the window frame will be displayed.

##### Usage:

- Name: [FrameEnabled](#)
- Format: "[text]".

##### Where [Text] is:

- "True" to indicate the windows frame should be displayed.
- "False" to indicate the windows frame should not be displayed.



## 6.101.2 Member Function Documentation

### 6.101.2.1 String CEGUI::FrameWindowProperties::FrameEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.101.2.2 void CEGUI::FrameWindowProperties::FrameEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

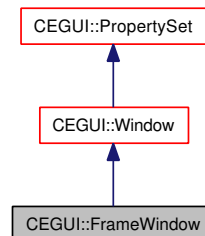
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

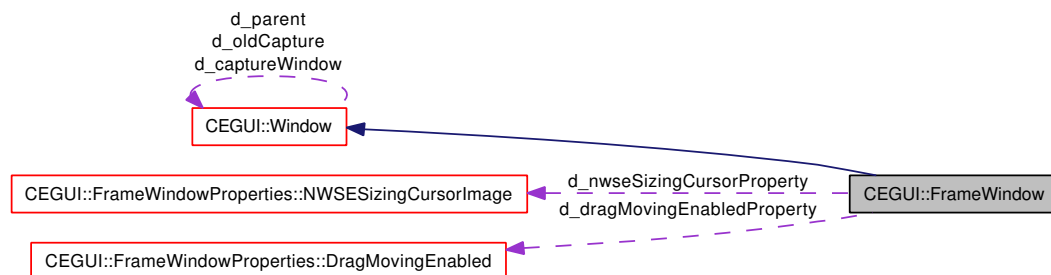
## 6.102 CEGUI::FrameWindow Class Reference

Abstract base class for a movable, sizable, window with a title-bar and a frame.

Inheritance diagram for CEGUI::FrameWindow:



Collaboration diagram for CEGUI::FrameWindow:



### Public Types

- enum [SizingLocation](#) {  
[SizingNone](#), [SizingTopLeft](#), [SizingTopRight](#), [SizingBottomLeft](#),  
[SizingBottomRight](#), [SizingTop](#), [SizingLeft](#), [SizingBottom](#),  
[SizingRight](#) }

*Enumeration that defines the set of possible locations for the mouse on a frame windows sizing border.*

### Public Member Functions

- virtual void [initialiseComponents](#) (void)  
*Initialises the [Window](#) based object ready for use.*
- bool [isSizingEnabled](#) (void) const  
*Return whether this window is sizable. Note that this requires that the window have an enabled frame and that sizing itself is enabled.*
- bool [isFrameEnabled](#) (void) const  
*Return whether the frame for this window is enabled.*

- bool [isTitleBarEnabled](#) (void) const  
*Return whether the title bar for this window is enabled.*
- bool [isCloseButtonEnabled](#) (void) const  
*Return whether this close button for this window is enabled.*
- bool [isRollupEnabled](#) (void) const  
*Return whether roll up (a.k.a shading) is enabled for this window.*
- bool [isRolledup](#) (void) const  
*Return whether the window is currently rolled up (a.k.a shaded).*
- float [getSizingBorderThickness](#) (void) const  
*Return the thickness of the sizing border.*
- void [setSizingEnabled](#) (bool setting)  
*Enables or disables sizing for this window.*
- void [setFrameEnabled](#) (bool setting)  
*Enables or disables the frame for this window.*
- void [setTitleBarEnabled](#) (bool setting)  
*Enables or disables the title bar for the frame window.*
- void [setCloseButtonEnabled](#) (bool setting)  
*Enables or disables the close button for the frame window.*
- void [setRollupEnabled](#) (bool setting)  
*Enables or disables roll-up (shading) for this window.*
- void [toggleRollup](#) (void)  
*Toggles the state of the window between rolled-up (shaded) and normal sizes. This requires roll-up to be enabled.*
- void [setSizingBorderThickness](#) (float pixels)  
*Set the size of the sizing border for this window.*
- void [offsetPixelPosition](#) (const [Vector2](#) &offset)  
*Move the window by the pixel offsets specified in offset.*
- bool [isDragMovingEnabled](#) (void) const  
*Return whether this [FrameWindow](#) can be moved by dragging the title bar.*
- void [setDragMovingEnabled](#) (bool setting)  
*Set whether this [FrameWindow](#) can be moved by dragging the title bar.*
- const [Image](#) \* [getNSSizingCursorImage](#) () const  
*Return a pointer to the currently set [Image](#) to be used for the north-south sizing mouse cursor.*
- const [Image](#) \* [getEWSizingCursorImage](#) () const

Return a pointer to the currently set *Image* to be used for the east-west sizing mouse cursor.

- `const Image * getNWSEizingCursorImage () const`

Return a pointer to the currently set *Image* to be used for the northwest-southeast sizing mouse cursor.

- `const Image * getNESWSizingCursorImage () const`

Return a pointer to the currently set *Image* to be used for the northeast-southwest sizing mouse cursor.

- `void setNSSizingCursorImage (const Image *image)`

Set the *Image* to be used for the north-south sizing mouse cursor.

- `void setEWSizingCursorImage (const Image *image)`

Set the *Image* to be used for the east-west sizing mouse cursor.

- `void setNWSEizingCursorImage (const Image *image)`

Set the *Image* to be used for the northwest-southeast sizing mouse cursor.

- `void setNESWSizingCursorImage (const Image *image)`

Set the *Image* to be used for the northeast-southwest sizing mouse cursor.

- `void setNSSizingCursorImage (const String &imageset, const String &image)`

Set the image to be used for the north-south sizing mouse cursor.

- `void setEWSizingCursorImage (const String &imageset, const String &image)`

Set the image to be used for the east-west sizing mouse cursor.

- `void setNWSEizingCursorImage (const String &imageset, const String &image)`

Set the image to be used for the northwest-southeast sizing mouse cursor.

- `void setNESWSizingCursorImage (const String &imageset, const String &image)`

Set the image to be used for the northeast-southwest sizing mouse cursor.

- `bool isHit (const Point &position) const`

check if the given pixel position would hit this window.

- `Titlebar * getTitlebar () const`

Return a pointer to the *Titlebar* component widget for this *FrameWindow*.

- `PushButton * getCloseButton () const`

Return a pointer to the close button component widget for this *FrameWindow*.

- `FrameWindow (const String &name, const String &type)`

Constructor for *FrameWindow* objects.

- `virtual ~FrameWindow (void)`

Destructor for *FramwWindow* objects.

## Static Public Attributes

- static const [String EventNamespace](#)  
*Namespace for global events.*
- static const [String WidgetTypeName](#)  
*Window factory name.*
- static const [String EventRollupToggled](#)  
*Fired when the rollup (shade) state of the window changes.*
- static const [String EventCloseClicked](#)  
*Fired when the close button for the window is clicked.*
- static const float [DefaultSizingBorderSize](#) = 8.0f  
*Default size for the sizing border (in pixels).*
- static const [String TitlebarNameSuffix](#)  
*Widget name suffix for the titlebar component.*
- static const [String CloseButtonNameSuffix](#)  
*Widget name suffix for the close button component.*

## Protected Member Functions

- void [moveLeftEdge](#) (float delta)  
*move the window's left edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).*
- void [moveRightEdge](#) (float delta)  
*move the window's right edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).*
- void [moveTopEdge](#) (float delta)  
*move the window's top edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).*
- void [moveBottomEdge](#) (float delta)  
*move the window's bottom edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).*
- [SizingLocation](#) [getSizingBorderAtPoint](#) (const [Point](#) &pt) const  
*check local pixel co-ordinate point 'pt' and return one of the [SizingLocation](#) enumerated values depending where the point falls on the sizing border.*
- bool [isLeftSizingLocation](#) ([SizingLocation](#) loc) const  
*return true if given [SizingLocation](#) is on left edge.*
- bool [isRightSizingLocation](#) ([SizingLocation](#) loc) const

*return true if given SizingLocation is on right edge.*

- bool `isTopSizingLocation` (`SizingLocation` loc) const  
*return true if given SizingLocation is on top edge.*
- bool `isBottomSizingLocation` (`SizingLocation` loc) const  
*return true if given SizingLocation is on bottom edge.*
- bool `closeClickHandler` (const `EventArgs` &e)  
*Method to respond to close button click events and fire our close event.*
- void `setCursorForPoint` (const `Point` &pt) const  
*Set the appropriate mouse cursor for the given window-relative pixel point.*
- virtual `Rect` `getSizingRect` (void) const  
*Return a Rect that describes, in window relative pixel co-ordinates, the outer edge of the sizing area for this window.*
- virtual bool `testClassName_impl` (const `String` &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void `onRollupToggled` (`WindowEventArgs` &e)  
*Event generated internally whenever the roll-up / shade state of the window changes.*
- virtual void `onCloseClicked` (`WindowEventArgs` &e)  
*Event generated internally whenever the close button is clicked.*
- virtual void `onMouseMove` (`MouseEventArgs` &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void `onMouseDown` (`MouseEventArgs` &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void `onMouseUp` (`MouseEventArgs` &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void `onCaptureLost` (`WindowEventArgs` &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void `onTextChanged` (`WindowEventArgs` &e)  
*Handler called when the window's text is changed.*
- virtual void `onActivated` (`ActivationEventArgs` &e)  
*Handler called when this window has become the active window.*
- virtual void `onDeactivated` (`ActivationEventArgs` &e)  
*Handler called when this window has lost input focus and has been deactivated.*

## Protected Attributes

- bool [d\\_frameEnabled](#)  
*true if window frame should be drawn.*
- bool [d\\_rollupEnabled](#)  
*true if roll-up of window is allowed.*
- bool [d\\_rolledup](#)  
*true if window is rolled up.*
- bool [d\\_sizingEnabled](#)  
*true if sizing is enabled for this window.*
- bool [d\\_beingSized](#)  
*true if window is being sized.*
- float [d\\_borderSize](#)  
*thickness of the sizing border around this window*
- [Point](#) [d\\_dragPoint](#)  
*point window is being dragged at.*
- const [Image](#) \* [d\\_nsSizingCursor](#)  
*North/South sizing cursor image.*
- const [Image](#) \* [d\\_ewSizingCursor](#)  
*East/West sizing cursor image.*
- const [Image](#) \* [d\\_nwseSizingCursor](#)  
*North-West/South-East cursor image.*
- const [Image](#) \* [d\\_neswSizingCursor](#)  
*North-East/South-West cursor image.*
- bool [d\\_dragMovable](#)  
*true if the window will move when dragged by the title bar.*

### 6.102.1 Detailed Description

Abstract base class for a movable, sizable, window with a title-bar and a frame.

### 6.102.2 Member Enumeration Documentation

#### 6.102.2.1 enum CEGUI::FrameWindow::SizingLocation

Enumeration that defines the set of possible locations for the mouse on a frame windows sizing border.

**Enumerator:**

- SizingNone* Position is not a sizing location.
- SizingTopLeft* Position will size from the top-left.
- SizingTopRight* Position will size from the top-right.
- SizingBottomLeft* Position will size from the bottom left.
- SizingBottomRight* Position will size from the bottom right.
- SizingTop* Position will size from the top.
- SizingLeft* Position will size from the left.
- SizingBottom* Position will size from the bottom.
- SizingRight* Position will size from the right.

### 6.102.3 Member Function Documentation

#### 6.102.3.1 void CEGUI::FrameWindow::initialiseComponents (void) [virtual]

Initialises the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

#### 6.102.3.2 bool CEGUI::FrameWindow::isSizingEnabled (void) const [inline]

Return whether this window is sizable. Note that this requires that the window have an enabled frame and that sizing itself is enabled.

**Returns:**

true if the window can be sized, false if the window can not be sized

#### 6.102.3.3 bool CEGUI::FrameWindow::isFrameEnabled (void) const [inline]

Return whether the frame for this window is enabled.

**Returns:**

true if the frame for this window is enabled, false if the frame for this window is disabled.



**6.102.3.4 bool CEGUI::FrameWindow::isTitleBarEnabled (void) const**

Return whether the title bar for this window is enabled.

**Returns:**

true if the window has a title bar and it is enabled, false if the window has no title bar or if the title bar is disabled.

**6.102.3.5 bool CEGUI::FrameWindow::isCloseButtonEnabled (void) const**

Return whether this close button for this window is enabled.

**Returns:**

true if the window has a close button and it is enabled, false if the window either has no close button or if the close button is disabled.

**6.102.3.6 bool CEGUI::FrameWindow::isRollupEnabled (void) const** [inline]

Return whether roll up (a.k.a shading) is enabled for this window.

**Returns:**

true if roll up is enabled, false if roll up is disabled.

**6.102.3.7 bool CEGUI::FrameWindow::isRolledup (void) const** [inline]

Return whether the window is currently rolled up (a.k.a shaded).

**Returns:**

true if the window is rolled up, false if the window is not rolled up.

**6.102.3.8 float CEGUI::FrameWindow::getSizingBorderThickness (void) const** [inline]

Return the thickness of the sizing border.

**Returns:**

float value describing the thickness of the sizing border in screen pixels.

**6.102.3.9 void CEGUI::FrameWindow::setSizingEnabled (bool *setting*)**

Enables or disables sizing for this window.

**Parameters:**

*setting* set to true to enable sizing (also requires frame to be enabled), or false to disable sizing.

**Returns:**

nothing

**6.102.3.10 void CEGUI::FrameWindow::setFrameEnabled (bool *setting*)**

Enables or disables the frame for this window.

**Parameters:**

*setting* set to true to enable the frame for this window, or false to disable the frame for this window.

**Returns:**

Nothing.

**6.102.3.11 void CEGUI::FrameWindow::setTitleBarEnabled (bool *setting*)**

Enables or disables the title bar for the frame window.

**Parameters:**

*setting* set to true to enable the title bar (if one is attached), or false to disable the title bar.

**Returns:**

Nothing.

**6.102.3.12 void CEGUI::FrameWindow::setCloseButtonEnabled (bool *setting*)**

Enables or disables the close button for the frame window.

**Parameters:**

*setting* Set to true to enable the close button (if one is attached), or false to disable the close button.

**Returns:**

Nothing.

**6.102.3.13 void CEGUI::FrameWindow::setRollupEnabled (bool *setting*)**

Enables or disables roll-up (shading) for this window.

**Parameters:**

*setting* Set to true to enable roll-up for the frame window, or false to disable roll-up.

**Returns:**

Nothing.

**6.102.3.14 void CEGUI::FrameWindow::toggleRollup (void)**

Toggles the state of the window between rolled-up (shaded) and normal sizes. This requires roll-up to be enabled.

**Returns:**

Nothing

**6.102.3.15 void CEGUI::FrameWindow::setSizeingBorderThickness (float *pixels*) [inline]**

Set the size of the sizing border for this window.

**Parameters:**

*pixels* float value specifying the thickness for the sizing border in screen pixels.

**Returns:**

Nothing.

**6.102.3.16 void CEGUI::FrameWindow::offsetPixelPosition (const Vector2 & *offset*)**

Move the window by the pixel offsets specified in *offset*.

This is intended for internal system use - it is the method by which the title bar moves the frame window.

**Parameters:**

*offset* [Vector2](#) object containing the offsets to apply (offsets are in screen pixels).

**Returns:**

Nothing.

**6.102.3.17 bool CEGUI::FrameWindow::isDragMovingEnabled (void) const [inline]**

Return whether this [FrameWindow](#) can be moved by dragging the title bar.

**Returns:**

true if the [Window](#) will move when the user drags the title bar, false if the window will not move.

**6.102.3.18 void CEGUI::FrameWindow::setDragMovingEnabled (bool *setting*)**

Set whether this [FrameWindow](#) can be moved by dragging the title bar.

**Parameters:**

*setting* true if the [Window](#) should move when the user drags the title bar, false if the window should not move.

**Returns:**

Nothing.

**6.102.3.19 const Image \* CEGUI::FrameWindow::getNSSizingCursorImage () const**

Return a pointer to the currently set [Image](#) to be used for the north-south sizing mouse cursor.

**Returns:**

Pointer to an [Image](#) object, or 0 for none.

**6.102.3.20** `const Image * CEGUI::FrameWindow::getEWSizingCursorImage () const`

Return a pointer to the currently set [Image](#) to be used for the east-west sizing mouse cursor.

**Returns:**

Pointer to an [Image](#) object, or 0 for none.

**6.102.3.21** `const Image * CEGUI::FrameWindow::getNWSEsizingCursorImage () const`

Return a pointer to the currently set [Image](#) to be used for the northwest-southeast sizing mouse cursor.

**Returns:**

Pointer to an [Image](#) object, or 0 for none.

**6.102.3.22** `const Image * CEGUI::FrameWindow::getNESWSizingCursorImage () const`

Return a pointer to the currently set [Image](#) to be used for the northeast-southwest sizing mouse cursor.

**Returns:**

Pointer to an [Image](#) object, or 0 for none.

**6.102.3.23** `void CEGUI::FrameWindow::setNSSizingCursorImage (const Image * image)`

Set the [Image](#) to be used for the north-south sizing mouse cursor.

**Parameters:**

*image* Pointer to an [Image](#) object, or 0 for none.

**Returns:**

Nothing.

**6.102.3.24** `void CEGUI::FrameWindow::setEWSizingCursorImage (const Image * image)`

Set the [Image](#) to be used for the east-west sizing mouse cursor.

**Parameters:**

*image* Pointer to an [Image](#) object, or 0 for none.

**Returns:**

Nothing.

**6.102.3.25 void CEGUI::FrameWindow::setNWSEizingCursorImage (const Image \* *image*)**

Set the [Image](#) to be used for the northwest-southeast sizing mouse cursor.

**Parameters:**

*image* Pointer to an [Image](#) object, or 0 for none.

**Returns:**

Nothing.

**6.102.3.26 void CEGUI::FrameWindow::setNESWSizingCursorImage (const Image \* *image*)**

Set the [Image](#) to be used for the northeast-southwest sizing mouse cursor.

**Parameters:**

*image* Pointer to an [Image](#) object, or 0 for none.

**Returns:**

Nothing.

**6.102.3.27 void CEGUI::FrameWindow::setNSSizingCursorImage (const String & *imageset*, const String & *image*)**

Set the image to be used for the north-south sizing mouse cursor.

**Parameters:**

*imageset* [String](#) holding the name of the [Imageset](#) containing the [Image](#) to be used.

*image* [String](#) holding the name of the [Image](#) to be used.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if either *imageset* or *image* refer to non-existent entities.

**6.102.3.28 void CEGUI::FrameWindow::setEWSizingCursorImage (const String & *imageset*, const String & *image*)**

Set the image to be used for the east-west sizing mouse cursor.

**Parameters:**

*imageset* [String](#) holding the name of the [Imageset](#) containing the [Image](#) to be used.

*image* [String](#) holding the name of the [Image](#) to be used.

**Returns:**

Nothing.

**Exceptions:**

*UnknownObjectException* thrown if either *imageset* or *image* refer to non-existent entities.

**6.102.3.29 void CEGUI::FrameWindow::setNWSE sizingCursorImage (const String & imageset, const String & image)**

Set the image to be used for the northwest-southeast sizing mouse cursor.

**Parameters:**

*imageset* *String* holding the name of the *Imageset* containing the *Image* to be used.

*image* *String* holding the name of the *Image* to be used.

**Returns:**

Nothing.

**Exceptions:**

*UnknownObjectException* thrown if either *imageset* or *image* refer to non-existent entities.

**6.102.3.30 void CEGUI::FrameWindow::setNESWSizingCursorImage (const String & imageset, const String & image)**

Set the image to be used for the northeast-southwest sizing mouse cursor.

**Parameters:**

*imageset* *String* holding the name of the *Imageset* containing the *Image* to be used.

*image* *String* holding the name of the *Image* to be used.

**Returns:**

Nothing.

**Exceptions:**

*UnknownObjectException* thrown if either *imageset* or *image* refer to non-existent entities.

**6.102.3.31 bool CEGUI::FrameWindow::isHit (const Point & position) const** [*inline*, *virtual*]

check if the given pixel position would hit this window.

**Parameters:**

*position* *Vector2* object describing the position to check. The position describes a pixel offset from the top-left corner of the display.

**Returns:**

- true if *position* hits this [Window](#).
- false if *position* does not hit this window.

Reimplemented from [CEGUI::Window](#).

**6.102.3.32 Titlebar \* CEGUI::FrameWindow::getTitlebar () const**

Return a pointer to the [Titlebar](#) component widget for this [FrameWindow](#).

**Returns:**

Pointer to a [Titlebar](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the [Titlebar](#) component does not exist.

**6.102.3.33 PushButton \* CEGUI::FrameWindow::getCloseButton () const**

Return a pointer to the close button component widget for this [FrameWindow](#).

**Returns:**

Pointer to a [PushButton](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the close button component does not exist.

**6.102.3.34 void CEGUI::FrameWindow::moveLeftEdge (float *delta*) [protected]**

move the window's left edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).

**Parameters:**

*delta* float value that specifies the amount to move the window edge, and in which direction. Positive values make window smaller.

**6.102.3.35 void CEGUI::FrameWindow::moveRightEdge (float *delta*) [protected]**

move the window's right edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).

**Parameters:**

*delta* float value that specifies the amount to move the window edge, and in which direction. Positive values make window larger.

**6.102.3.36 void CEGUI::FrameWindow::moveTopEdge (float *delta*) [protected]**

move the window's top edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).

**Parameters:**

*delta* float value that specifies the amount to move the window edge, and in which direction. Positive values make window smaller.

**6.102.3.37 void CEGUI::FrameWindow::moveBottomEdge (float *delta*) [protected]**

move the window's bottom edge by 'delta'. The rest of the window does not move, thus this changes the size of the [Window](#).

**Parameters:**

*delta* float value that specifies the amount to move the window edge, and in which direction. Positive values make window larger.

**6.102.3.38 FrameWindow::SizingLocation CEGUI::FrameWindow::getSizingBorderAtPoint (const Point & *pt*) const [protected]**

check local pixel co-ordinate point 'pt' and return one of the SizingLocation enumerated values depending where the point falls on the sizing border.

**Parameters:**

*pt* Point object describing, in pixels, the window relative offset to check.

**Returns:**

One of the SizingLocation enumerated values that describe which part of the sizing border that *pt* corresponded to, if any.

**6.102.3.39 bool CEGUI::FrameWindow::isLeftSizingLocation (SizingLocation *loc*) const [inline, protected]**

return true if given SizingLocation is on left edge.

**Parameters:**

*loc* SizingLocation value to be checked.

**Returns:**

true if *loc* is on the left edge. false if *loc* is not on the left edge.



**6.102.3.40 bool CEGUI::FrameWindow::isRightSizingLocation (SizingLocation *loc*) const**  
[inline, protected]

return true if given SizingLocation is on right edge.

**Parameters:**

*loc* SizingLocation value to be checked.

**Returns:**

true if *loc* is on the right edge. false if *loc* is not on the right edge.

**6.102.3.41 bool CEGUI::FrameWindow::isTopSizingLocation (SizingLocation *loc*) const**  
[inline, protected]

return true if given SizingLocation is on top edge.

**Parameters:**

*loc* SizingLocation value to be checked.

**Returns:**

true if *loc* is on the top edge. false if *loc* is not on the top edge.

**6.102.3.42 bool CEGUI::FrameWindow::isBottomSizingLocation (SizingLocation *loc*) const**  
[inline, protected]

return true if given SizingLocation is on bottom edge.

**Parameters:**

*loc* SizingLocation value to be checked.

**Returns:**

true if *loc* is on the bottom edge. false if *loc* is not on the bottom edge.

**6.102.3.43 virtual bool CEGUI::FrameWindow::testClassName\_impl (const String & *class\_name*) const**  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.102.3.44** `void CEGUI::FrameWindow::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.102.3.45** `void CEGUI::FrameWindow::onMouseButtonDown (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.102.3.46** `void CEGUI::FrameWindow::onMouseButtonUp (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.102.3.47** `void CEGUI::FrameWindow::onCaptureLost (WindowEventArgs & e)`  
[protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.102.3.48** `void CEGUI::FrameWindow::onTextChanged (WindowEventArgs & e)`  
[protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.102.3.49 void CEGUI::FrameWindow::onActivated (ActivationEventArgs & *e*)**  
[protected, virtual]

Handler called when this window has become the active window.

**Parameters:**

- e* [ActivationEventArgs](#) class whose 'otherWindow' field is set to the window that previously was active, or NULL for none.

Reimplemented from [CEGUI::Window](#).

**6.102.3.50 void CEGUI::FrameWindow::onDeactivated (ActivationEventArgs & *e*)**  
[protected, virtual]

Handler called when this window has lost input focus and has been deactivated.

**Parameters:**

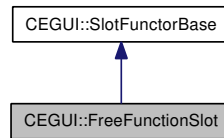
- e* [ActivationEventArgs](#) object whose 'otherWindow' field is set to the window that has now become active, or NULL for none.

Reimplemented from [CEGUI::Window](#).

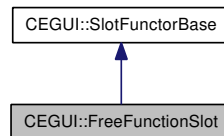
## 6.103 CEGUI::FreeFunctionSlot Class Reference

Slot functor class that calls back via a free function pointer.

Inheritance diagram for CEGUI::FreeFunctionSlot:



Collaboration diagram for CEGUI::FreeFunctionSlot:



### Public Member Functions

- typedef bool() [SlotFunction](#) (const [EventArgs](#) &)  
*Slot function type.*
- **FreeFunctionSlot** (SlotFunction \*func)
- virtual bool **operator()** (const [EventArgs](#) &args)

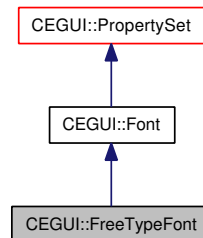
### 6.103.1 Detailed Description

Slot functor class that calls back via a free function pointer.

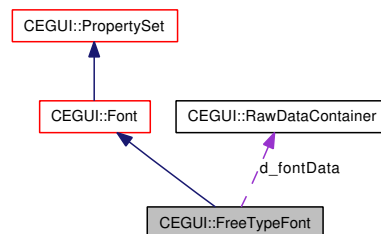
## 6.104 CEGUI::FreeTypeFont Class Reference

Implementation of the [Font](#) class interface using the FreeType library.

Inheritance diagram for CEGUI::FreeTypeFont:



Collaboration diagram for CEGUI::FreeTypeFont:



### Public Member Functions

- virtual void [load](#) ()

*Complete font loading. If you create the font from an XML file, this method is invoked automatically after reading all the required data from the [XMLAttributes](#) object. If you create the font manually, it is your responsibility to call this function as soon as you set up all the appropriate fields of the [Font](#) object.*

- virtual void [rasterize](#) (utf32 start\_codepoint, utf32 end\_codepoint)

*This function prepares a certain range of glyphs to be ready for displaying. This means that after returning from this function glyphs from `d_cp_map[start_codepoint]` to `d_cp_map[end_codepoint]` should have their `d_image` member set. If there is an error during rasterization of some glyph, it's okay to leave the `d_image` field set to NULL, in which case such glyphs will be skipped from display.*

**Parameters:**

***start\_codepoint** The lowest codepoint that should be rasterized*  
***end\_codepoint** The highest codepoint that should be rasterized*

### Protected Types

- typedef std::vector< [Imageset](#) \* > **ImagesetVector**

### Protected Member Functions

- void [drawGlyphToBuffer](#) ([argb\\_t](#) \*buffer, uint buf\_width)

Copy the current glyph data into buffer, which has a width of buf\_width pixels (not bytes).

- uint [getTexturSize](#) (CodepointMap::const\_iterator s, CodepointMap::const\_iterator e)

Return the required texture size required to store imagery for the glyphs from s to e.

- [FreeTypeFont](#) (const [String](#) &name, const [String](#) &filename, const [String](#) &resourceGroup)

Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to set up all remaining font parameters after constructing the [Font](#) object, and finally calling the [load\(\)](#) method which will make font available for use. All font parameters that are not initialized are set to sensible default values.

**Parameters:**

**name** The unique name that will be used to identify this [Font](#).

**fontname** The filename of the font file, which contains the font data. This can be a TrueType, PostScript, bitmap font etc file.

**resourceGroup** Resource group identifier to be passed to the resource provider to load the font definition file.

- [FreeTypeFont](#) (const [XMLAttributes](#) &attributes)

Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to set up all remaining font parameters after constructing the [Font](#) object, and finally calling the [load\(\)](#) method which will make font available for use. All font parameters that are not initialized are set to sensible default values.

**Parameters:**

**name** The unique name that will be used to identify this [Font](#).

**fontname** The filename of the font file, which contains the font data. This can be a TrueType, PostScript, bitmap font etc file.

**resourceGroup** Resource group identifier to be passed to the resource provider to load the font definition file.

- virtual [~FreeTypeFont](#) ()

Destroys a [Font](#) object.

- virtual void [updateFont](#) ()

Update the font as required according to the current parameters.

- virtual void [writeXMLToStream\\_impl](#) ([XMLSerializer](#) &xml\_stream) const

Same as [writeXMLToStream\(\)](#) but called from inside [writeXMLToStream\(\)](#) so that derived classes may add their own attributes to stream.

**Parameters:**

**xml\_stream** Stream where xml data should be output.

- void [addFreeTypeFontProperties](#) ()

Register all properties of this class.

- void [free](#) ()

Free all allocated font data.

## Protected Attributes

- [ImagesetVector](#) [d\\_glyphImages](#)

Imagesets that holds the glyphs for this font.

- float [d\\_ptSize](#)

*Point size of font.*

- bool [d\\_antiAliased](#)  
*True if the font should be rendered as anti-aliased by freeType.*
- FT\_Face [d\\_fontFace](#)  
*FreeType-specific font handle.*
- RawDataContainer [d\\_fontData](#)  
*Font file data.*

## Friends

- class **FontManager**
- class **FontProperties::FreeTypePointSize**
- class **FontProperties::FreeTypeAntialiased**

## 6.104.1 Detailed Description

Implementation of the [Font](#) class interface using the FreeType library.

This implementation tries to provide maximal support for any kind of fonts supported by FreeType. It has been tested on outline font formats like TTF and PS as well as on bitmap font formats like PCF and FON.

Glyphs are rendered dynamically on demand, so a large font with lots of glyphs won't slow application startup time.

## 6.104.2 Member Function Documentation

**6.104.2.1** void CEGUI::FreeTypeFont::drawGlyphToBuffer (argb\_t \* *buffer*, uint *buf\_width*)  
[protected]

Copy the current glyph data into *buffer*, which has a width of *buf\_width* pixels (not bytes).

### Parameters:

- buffer* Memory buffer large enough to receive the imagery for the currently loaded glyph.
- buf\_width* Width of *buffer* in pixels (where each pixel is a argb\_t).

### Returns:

Nothing.

**6.104.2.2** uint CEGUI::FreeTypeFont::getTextureSize (CodepointMap::const\_iterator *s*, CodepointMap::const\_iterator *e*) [protected]

Return the required texture size required to store imagery for the glyphs from *s* to *e*.

### Parameters:

- s* The first glyph in set

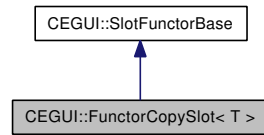
*e* The last glyph in set



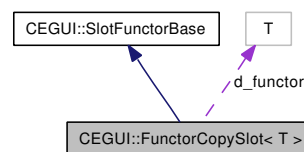
## 6.105 CEGUI::FunctorCopySlot< T > Class Template Reference

Slot template class that creates a functor that calls back via a copy of a functor object.

Inheritance diagram for CEGUI::FunctorCopySlot< T >:



Collaboration diagram for CEGUI::FunctorCopySlot< T >:



### Public Member Functions

- **FunctorCopySlot** (const T &functor)
- virtual bool **operator()** (const [EventArgs](#) &args)

### 6.105.1 Detailed Description

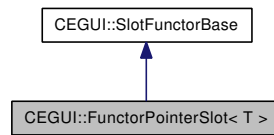
**template<typename T> class CEGUI::FunctorCopySlot< T >**

Slot template class that creates a functor that calls back via a copy of a functor object.

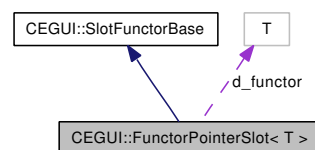
## 6.106 CEGUI::FunctorPointerSlot< T > Class Template Reference

Slot template class that creates a functor that calls back via a functor object pointer.

Inheritance diagram for CEGUI::FunctorPointerSlot< T >:



Collaboration diagram for CEGUI::FunctorPointerSlot< T >:



### Public Member Functions

- **FunctorPointerSlot** (T \*functor)
- virtual bool **operator()** (const [EventArgs](#) &args)

#### 6.106.1 Detailed Description

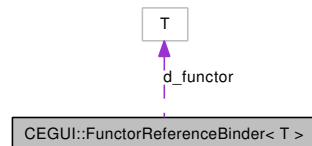
**template<typename T> class CEGUI::FunctorPointerSlot< T >**

Slot template class that creates a functor that calls back via a functor object pointer.

## 6.107 CEGUI::FunctorReferenceBinder< T > Struct Template Reference

Class that enables the creation of a reference binding for a functor object to be used as a callback slot. Wrap your functor with this to enable the use of an object reference when subscribing to an event signal (as opposed to the functor object being copied, or using a pointer).

Collaboration diagram for CEGUI::FunctorReferenceBinder< T >:



### Public Member Functions

- **FunctorReferenceBinder** (T &functor)

### Public Attributes

- T & **d\_functor**

### 6.107.1 Detailed Description

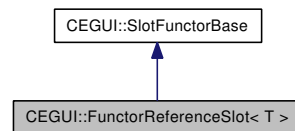
**template<typename T> struct CEGUI::FunctorReferenceBinder< T >**

Class that enables the creation of a reference binding for a functor object to be used as a callback slot. Wrap your functor with this to enable the use of an object reference when subscribing to an event signal (as opposed to the functor object being copied, or using a pointer).

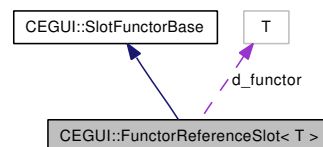
## 6.108 CEGUI::FunctorReferenceSlot< T > Class Template Reference

Slot template class that creates a functor that calls back via a functor object reference.

Inheritance diagram for CEGUI::FunctorReferenceSlot< T >:



Collaboration diagram for CEGUI::FunctorReferenceSlot< T >:



### Public Member Functions

- **FunctorReferenceSlot** (T &functor)
- virtual bool **operator()** (const [EventArgs](#) &args)

#### 6.108.1 Detailed Description

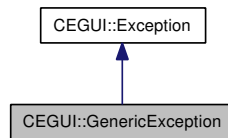
**template<typename T> class CEGUI::FunctorReferenceSlot< T >**

Slot template class that creates a functor that calls back via a functor object reference.

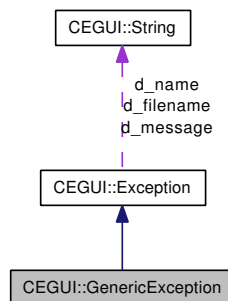
## 6.109 CEGUI::GenericException Class Reference

[Exception](#) class used when none of the other classes are applicable.

Inheritance diagram for CEGUI::GenericException:



Collaboration diagram for CEGUI::GenericException:



### Public Member Functions

- [GenericException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the generic exception by calling the base class.*

### 6.109.1 Detailed Description

[Exception](#) class used when none of the other classes are applicable.

### 6.109.2 Constructor & Destructor Documentation

#### 6.109.2.1 CEGUI::GenericException::GenericException (const String &message, const String &file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the generic exception by calling the base class.

#### Parameters:

- message* [String](#) object describing the reason for the generic exception being thrown.
- filename* [String](#) object containing the name of the file where the generic exception occurred.
- line* Integer representing the line number where the generic exception occurred.

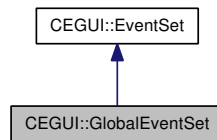
**Remarks:**

The generic exception name is automatically passed to the base class as "CEGUI::GenericException".

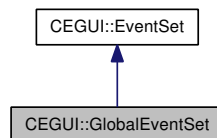
## 6.110 CEGUI::GlobalEventSet Class Reference

The [GlobalEventSet](#) singleton allows you to subscribe to an event for all instances of a class. The [GlobalEventSet](#) effectively supports "late binding" to events; which means you can subscribe to some event that does not actually exist (yet).

Inheritance diagram for CEGUI::GlobalEventSet:



Collaboration diagram for CEGUI::GlobalEventSet:



### Public Member Functions

- virtual void [fireEvent](#) (const [String](#) &name, [EventArgs](#) &args, const [String](#) &eventNamespace="")  
*Fires the named event passing the given [EventArgs](#) object.*

### Static Public Member Functions

- static [GlobalEventSet](#) & [getSingleton](#) (void)  
*Return singleton [System](#) object.*
- static [GlobalEventSet](#) \* [getSingletonPtr](#) (void)  
*Return pointer to singleton [System](#) object.*

#### 6.110.1 Detailed Description

The [GlobalEventSet](#) singleton allows you to subscribe to an event for all instances of a class. The [GlobalEventSet](#) effectively supports "late binding" to events; which means you can subscribe to some event that does not actually exist (yet).

#### 6.110.2 Member Function Documentation

##### 6.110.2.1 [GlobalEventSet](#) & CEGUI::GlobalEventSet::getSingleton (void) [static]

Return singleton [System](#) object.

**Returns:**

Singleton [System](#) object

**6.110.2.2** `GlobalEventSet * CEGUI::GlobalEventSet::getSingletonPtr (void)` `[static]`

Return pointer to singleton [System](#) object.

**Returns:**

Pointer to singleton [System](#) object

**6.110.2.3** `void CEGUI::GlobalEventSet::fireEvent (const String & name, EventArgs & args, const String & eventNamespace = "")` `[virtual]`

Fires the named event passing the given [EventArgs](#) object.

**Parameters:**

*name* [String](#) object holding the name of the [Event](#) that is to be fired (triggered)

*args* The [EventArgs](#) (or derived) object that is to be passed to each subscriber of the [Event](#). Once all subscribers have been called the 'handled' field of the event is updated appropriately.

*eventNamespace* [String](#) object describing the namespace prefix to use when firing the global event.

**Returns:**

Nothing.

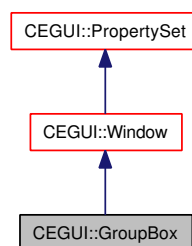
Reimplemented from [CEGUI::EventSet](#).



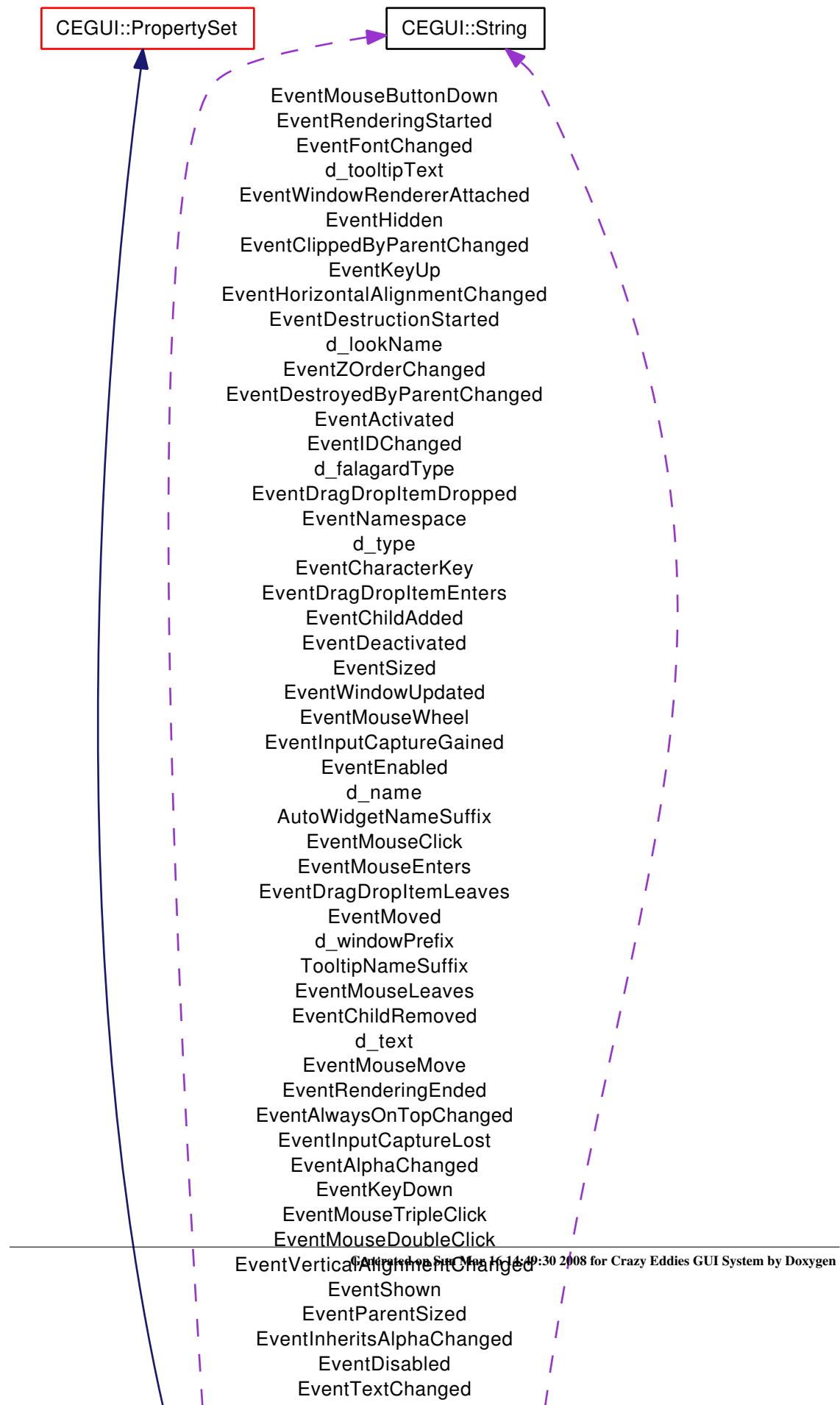
## 6.111 CEGUI::GroupBox Class Reference

Base class for standard [GroupBox](#) widget.

Inheritance diagram for CEGUI::GroupBox:



Collaboration diagram for CEGUI::GroupBox:



## Public Member Functions

- **GroupBox** (const [String](#) &type, const [String](#) &name)  
*Constructor for [GroupBox](#) class.*
- virtual **~GroupBox** ()  
*Destructor for [GroupBox](#) class.*
- bool **drawAroundWidget** (const [CEGUI::Window](#) \*wnd)  
*Draws the [GroupBox](#) around a widget. The size and position of the [GroupBox](#) are overridden. Once the window that is drawn around resizes, you'll have to call the function again. *FIXME*.*
- bool **drawAroundWidget** (const [String](#) &name)
- virtual bool **testClassName\_impl** (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- [Window](#) \* **getContentPane** () const  
*Returns the content pane held by this [GroupBox](#).*

## Static Public Attributes

- static const [String](#) **EventNamespace**  
*Namespace for global events.*
- static const [String](#) **WidgetTypeName**
- static const [String](#) **ContentPaneNameSuffix**

## Protected Member Functions

- virtual void **initialiseComponents** ()  
*Initializes the components necessary.*
- virtual void **addChild\_impl** ([Window](#) \*wnd)  
*Add given window to child list at an appropriate position.*
- virtual void **removeChild\_impl** ([Window](#) \*wnd)  
*Remove our child again when necessary.*

### 6.111.1 Detailed Description

Base class for standard [GroupBox](#) widget.

## 6.111.2 Member Function Documentation

### 6.111.2.1 `virtual bool CEGUI::GroupBox::testClassName_impl (const String & class_name) const` [inline, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

### 6.111.2.2 `Window * CEGUI::GroupBox::getContentPane () const`

Returns the content pane held by this [GroupBox](#).

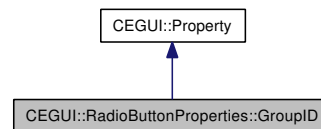
**Returns:**

Pointer to a [Window](#) instance.

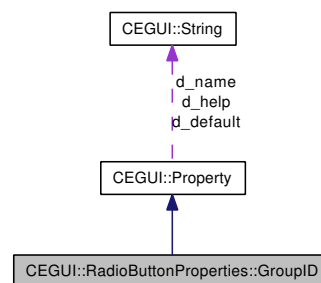
## 6.112 CEGUI::RadioButtonProperties::GroupID Class Reference

[Property](#) to access the radio button group ID.

Inheritance diagram for CEGUI::RadioButtonProperties::GroupID:



Collaboration diagram for CEGUI::RadioButtonProperties::GroupID:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.112.1 Detailed Description

[Property](#) to access the radio button group ID.

##### Usage:

- Name: [GroupID](#)
- Format: "[uint]".

##### Where:

- [uint] is any unsigned integer value.

## 6.112.2 Member Function Documentation

### 6.112.2.1 `String CEGUI::RadioButtonProperties::GroupID::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.112.2.2 `void CEGUI::RadioButtonProperties::GroupID::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

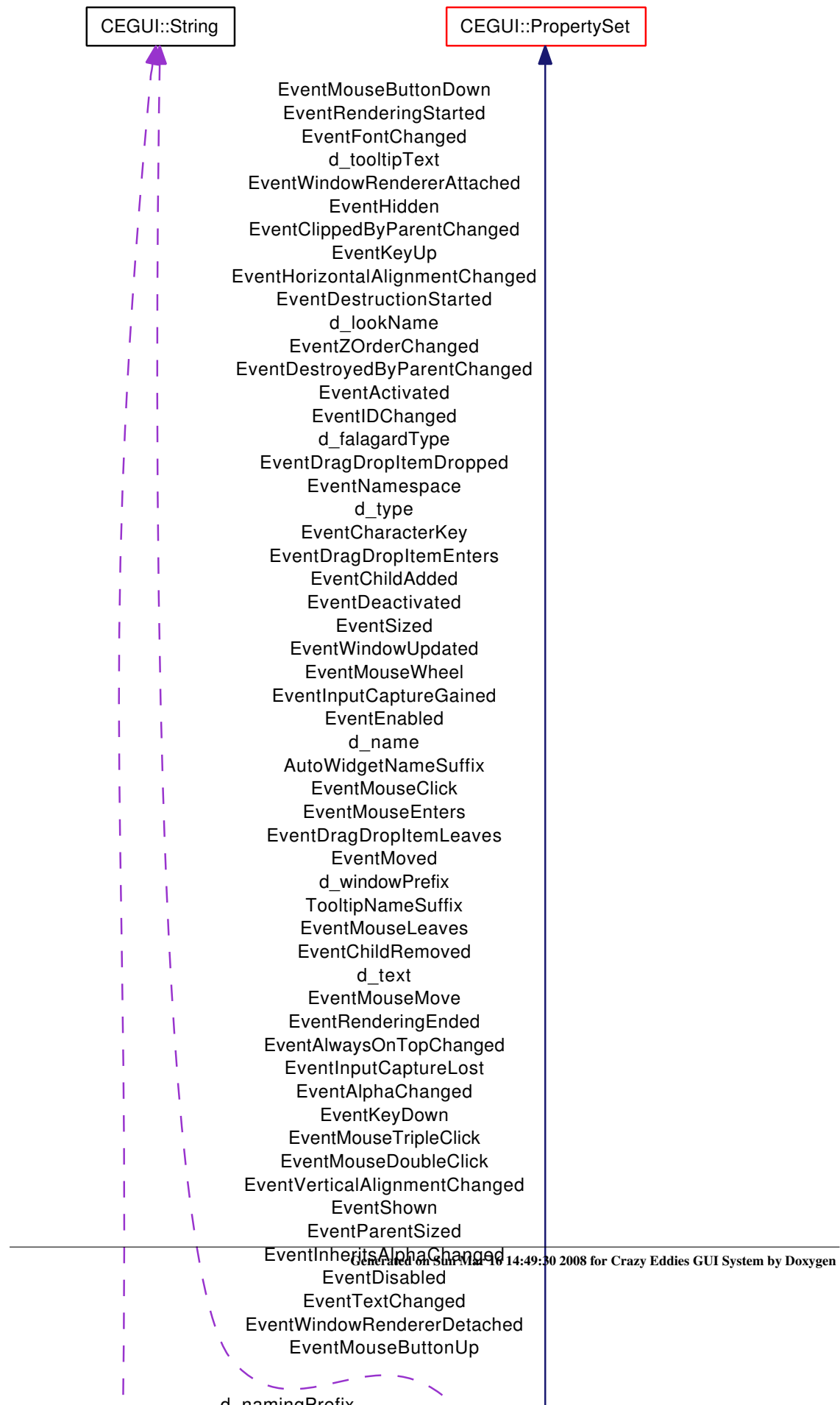
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.113 CEGUI::GUILayout\_xmlHandler Class Reference

Handler class used to parse the GUILayout XML files using SAX2.

Collaboration diagram for CEGUI::GUILayout\_xmlHandler:





## Public Member Functions

- **GUILayout\_xmlHandler** (const **String** &name\_prefix, PropertyCallback \*callback=0, void \*userdata=0)  
*Constructor for **GUILayout\_xmlHandler** objects.*
- virtual **~GUILayout\_xmlHandler** (void)  
*Destructor for **GUILayout\_xmlHandler** objects.*
- virtual void **elementStart** (const **String** &element, const **XMLAttributes** &attributes)  
*document processing (only care about elements, schema validates format)*
- virtual void **elementEnd** (const **String** &element)
- virtual void **text** (const **String** &text)
- void **cleanupLoadedWindows** (void)  
*Destroy all windows created so far.*
- **Window** \* **getLayoutRootWindow** (void) const  
*Return a pointer to the 'root' window created.*

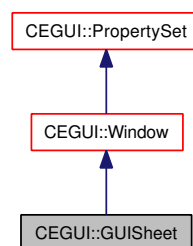
### 6.113.1 Detailed Description

Handler class used to parse the GUILayout XML files using SAX2.

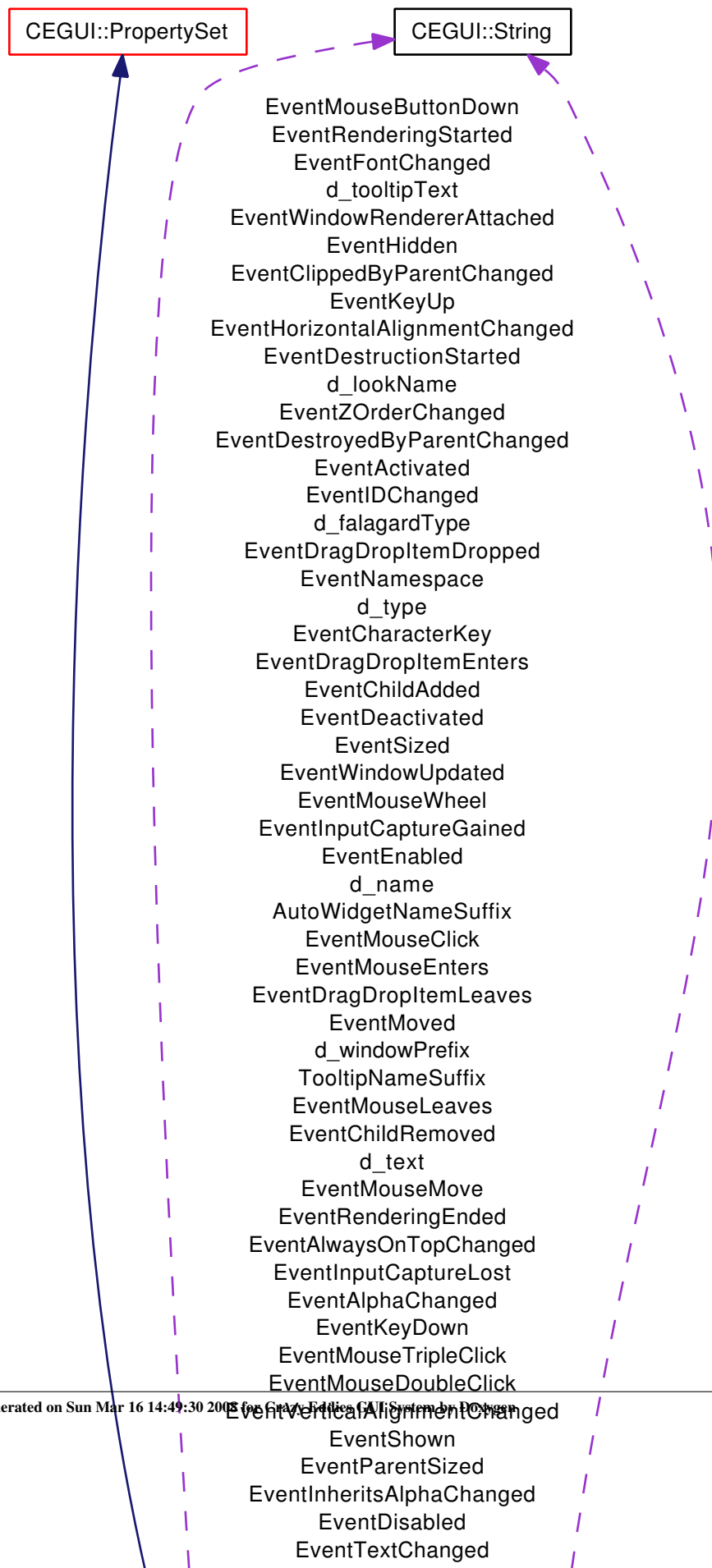
## 6.114 CEGUI::GUISheet Class Reference

[Window](#) class intended to be used as a simple, generic [Window](#).

Inheritance diagram for CEGUI::GUISheet:



Collaboration diagram for CEGUI::GUISheet:



## Public Member Functions

- [GUISheet](#) (const [String](#) &type, const [String](#) &name)

*Constructor for [GUISheet](#) windows.*

- virtual [~GUISheet](#) (void)

*Destructor for [GUISheet](#) windows.*

## Static Public Attributes

- static const [String](#) [WidgetTypeName](#)

*The unique typename of this widget.*

## Protected Member Functions

- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const

*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

### 6.114.1 Detailed Description

[Window](#) class intended to be used as a simple, generic [Window](#).

This class does no rendering and so appears totally transparent. This window defaults to position 0.0f, 0.0f with a size of 1.0f x 1.0f.

/note The C++ name of this class has been retained for backward compatibility reasons. The intended usage of this window type has now been extended beyond that of a gui-sheet or root window.

### 6.114.2 Member Function Documentation

#### 6.114.2.1 virtual bool CEGUI::GUISheet::testClassName\_impl (const [String](#) & *class\_name*) const [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

#### Parameters:

*class\_name* The class name that is to be checked.

#### Returns:

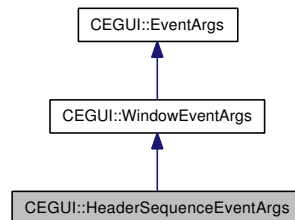
true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

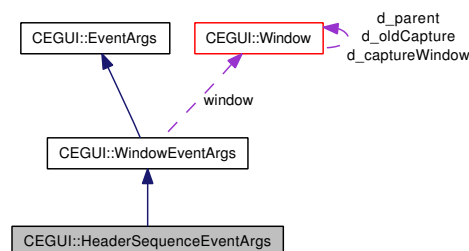
## 6.115 CEGUI::HeaderSequenceEventArgs Class Reference

[EventArgs](#) class used for segment move (sequence changed) events.

Inheritance diagram for CEGUI::HeaderSequenceEventArgs:



Collaboration diagram for CEGUI::HeaderSequenceEventArgs:



### Public Member Functions

- **HeaderSequenceEventArgs** ([Window](#) \*wnd, uint old\_idx, uint new\_idx)

### Public Attributes

- uint [d\\_oldIdx](#)  
*The original column index of the segment that has moved.*
- uint [d\\_newIdx](#)  
*The new column index of the segment that has moved.*

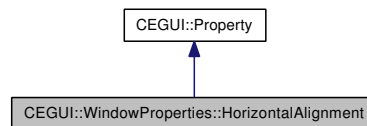
#### 6.115.1 Detailed Description

[EventArgs](#) class used for segment move (sequence changed) events.

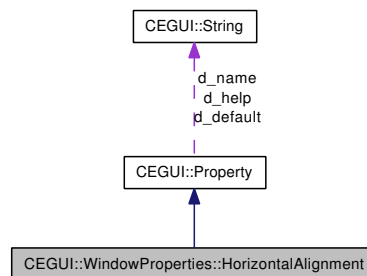
## 6.116 CEGUI::WindowProperties::HorizontalAlignment Class Reference

[Property](#) to access the horizontal alignment setting for the window.

Inheritance diagram for CEGUI::WindowProperties::HorizontalAlignment:



Collaboration diagram for CEGUI::WindowProperties::HorizontalAlignment:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.116.1 Detailed Description

[Property](#) to access the horizontal alignment setting for the window.

**Usage:**

- Name: [HorizontalAlignment](#)
- Format: "[text]".

**Where [Text] is:**

- "Left" to indicate the windows position is an offset of its left edge from its parents left edge.
- "Centre" to indicate the windows position is an offset of its centre point from its parents centre point.
- "Right" to indicate the windows position is an offset of its right edge from its parents right edge.

## 6.116.2 Member Function Documentation

### 6.116.2.1 String CEGUI::WindowProperties::HorizontalAlignment::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.116.2.2 void CEGUI::WindowProperties::HorizontalAlignment::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

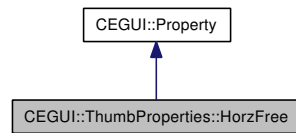
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

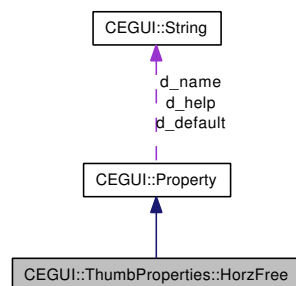
## 6.117 CEGUI::ThumbProperties::HorzFree Class Reference

[Property](#) to access the state the setting to free the thumb horizontally.

Inheritance diagram for CEGUI::ThumbProperties::HorzFree:



Collaboration diagram for CEGUI::ThumbProperties::HorzFree:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.117.1 Detailed Description

[Property](#) to access the state the setting to free the thumb horizontally.

Usage:

- Name: [HorzFree](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the thumb will be free (movable) horizontally.
- "False" to indicate the thumb will be fixed horizontally.



## 6.117.2 Member Function Documentation

### 6.117.2.1 String CEGUI::ThumbProperties::HorzFree::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.117.2.2 void CEGUI::ThumbProperties::HorzFree::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

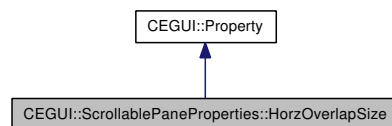
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

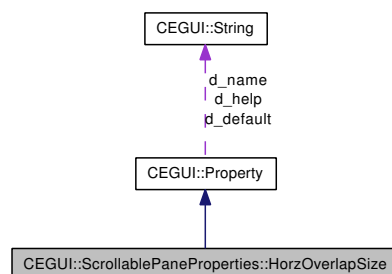
## 6.118 CEGUI::ScrollablePaneProperties::HorzOverlapSize Class Reference

[Property](#) to access the overlap size for the horizontal [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollablePaneProperties::HorzOverlapSize:



Collaboration diagram for CEGUI::ScrollablePaneProperties::HorzOverlapSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.118.1 Detailed Description

[Property](#) to access the overlap size for the horizontal [Scrollbar](#).

#### Usage:

- Name: [HorzOverlapSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the per-page overlap (as a fraction of one page).

## 6.118.2 Member Function Documentation

### 6.118.2.1 String CEGUI::ScrollablePaneProperties::HorzOverlapSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.118.2.2 void CEGUI::ScrollablePaneProperties::HorzOverlapSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

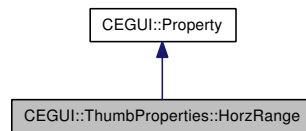
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

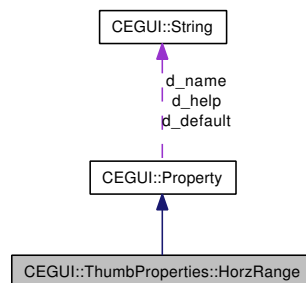
## 6.119 CEGUI::ThumbProperties::HorzRange Class Reference

[Property](#) to access the horizontal movement range for the thumb.

Inheritance diagram for CEGUI::ThumbProperties::HorzRange:



Collaboration diagram for CEGUI::ThumbProperties::HorzRange:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.119.1 Detailed Description

[Property](#) to access the horizontal movement range for the thumb.

#### Usage:

- Name: [HorzRange](#)
- Format: "min:[float] max:[float]".

#### Where:

- min:[float] specifies the minimum horizontal setting (position) for the thumb, specified using the active metrics system for the window.
- max:[float] specifies the maximum horizontal setting (position) for the thumb, specified using the active metrics system for the window.

## 6.119.2 Member Function Documentation

### 6.119.2.1 String CEGUI::ThumbProperties::HorzRange::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.119.2.2 void CEGUI::ThumbProperties::HorzRange::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

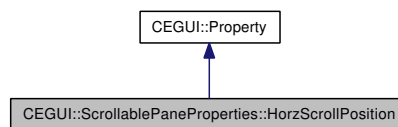
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

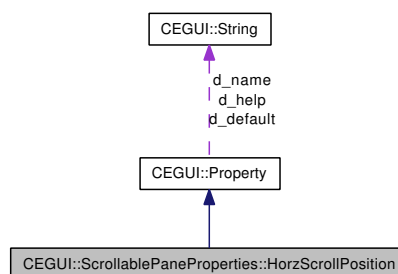
## 6.120 CEGUI::ScrollablePaneProperties::HorzScrollPosition Class Reference

[Property](#) to access the scroll position of the horizontal [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollablePaneProperties::HorzScrollPosition:



Collaboration diagram for CEGUI::ScrollablePaneProperties::HorzScrollPosition:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.120.1 Detailed Description

[Property](#) to access the scroll position of the horizontal [Scrollbar](#).

#### Usage:

- Name: [HorzScrollPosition](#)
- Format: "[float]".

#### Where:

- [float] specifies the current scroll position / value of the horizontal [Scrollbar](#) (as a fraction of the whole).

## 6.120.2 Member Function Documentation

### 6.120.2.1 String CEGUI::ScrollablePaneProperties::HorzScrollPosition::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.120.2.2 void CEGUI::ScrollablePaneProperties::HorzScrollPosition::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

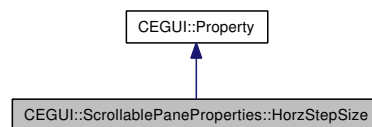
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

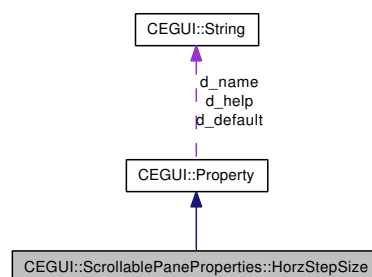
## 6.121 CEGUI::ScrollablePaneProperties::HorzStepSize Class Reference

[Property](#) to access the step size for the horizontal [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollablePaneProperties::HorzStepSize:



Collaboration diagram for CEGUI::ScrollablePaneProperties::HorzStepSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.121.1 Detailed Description

[Property](#) to access the step size for the horizontal [Scrollbar](#).

#### Usage:

- Name: [HorzStepSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the increase/decrease button step for the horizontal scrollbar (as a fraction of 1 page).



## 6.121.2 Member Function Documentation

### 6.121.2.1 String CEGUI::ScrollablePaneProperties::HorzStepSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.121.2.2 void CEGUI::ScrollablePaneProperties::HorzStepSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

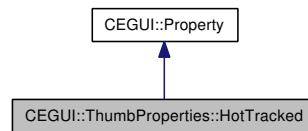
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

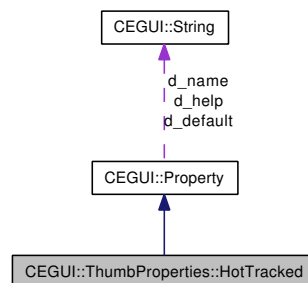
## 6.122 CEGUI::ThumbProperties::HotTracked Class Reference

[Property](#) to access the state of the "hot-tracked" setting for the thumb.

Inheritance diagram for CEGUI::ThumbProperties::HotTracked:



Collaboration diagram for CEGUI::ThumbProperties::HotTracked:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.122.1 Detailed Description

[Property](#) to access the state of the "hot-tracked" setting for the thumb.

Usage:

- Name: [HotTracked](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the thumb will send notifications as it is dragged.
- "False" to indicate the thumb will only notify once, when it is released.

## 6.122.2 Member Function Documentation

### 6.122.2.1 String CEGUI::ThumbProperties::HotTracked::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.122.2.2 void CEGUI::ThumbProperties::HotTracked::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

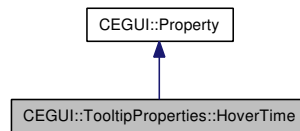
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

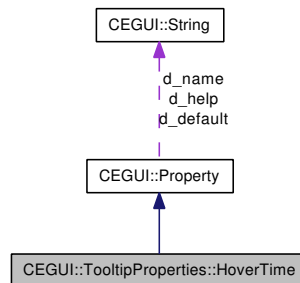
## 6.123 CEGUI::TooltipProperties::HoverTime Class Reference

[Property](#) to access the timeout that must expire before the tooltip gets activated.

Inheritance diagram for CEGUI::TooltipProperties::HoverTime:



Collaboration diagram for CEGUI::TooltipProperties::HoverTime:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.123.1 Detailed Description

[Property](#) to access the timeout that must expire before the tooltip gets activated.

#### Usage:

- Name: [HoverTime](#)
- Format: "[float]".

#### Where:

- [float] specifies the number of seconds the mouse must hover stationary on a widget before the tooltip gets activated.

## 6.123.2 Member Function Documentation

### 6.123.2.1 String CEGUI::TooltipProperties::HoverTime::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.123.2.2 void CEGUI::TooltipProperties::HoverTime::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

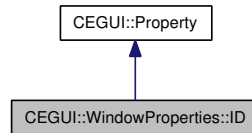
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

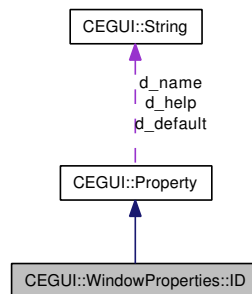
## 6.124 CEGUI::WindowProperties::ID Class Reference

[Property](#) to access window [ID](#) field.

Inheritance diagram for CEGUI::WindowProperties::ID:



Collaboration diagram for CEGUI::WindowProperties::ID:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.124.1 Detailed Description

[Property](#) to access window [ID](#) field.

This property offers access to the client specified [ID](#) for the window.

##### Usage:

- Name: [ID](#)
- Format: "[uint]".

##### Where:

- [uint] is any unsigned integer value.

## 6.124.2 Member Function Documentation

### 6.124.2.1 String CEGUI::WindowProperties::ID::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.124.2.2 void CEGUI::WindowProperties::ID::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

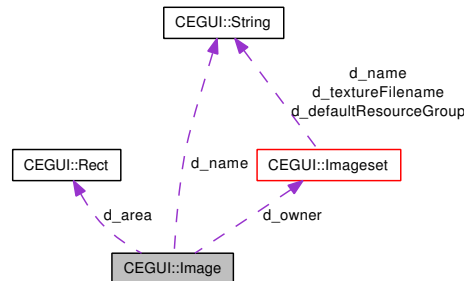
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.125 CEGUI::Image Class Reference

Class that represents a single [Image](#) of an [Imageset](#).

Collaboration diagram for CEGUI::Image:



### Public Member Functions

- [Size](#) `getSize` (void) const  
Return a [Size](#) object containing the dimensions of the [Image](#).
- float [getWidth](#) (void) const  
Return the pixel width of the image.
- float [getHeight](#) (void) const  
Return the pixel height of the image.
- [Point](#) `getOffsets` (void) const  
Return a [Point](#) object that contains the offset applied when rendering this [Image](#).
- float [getOffsetX](#) (void) const  
Return the X rendering offset.
- float [getOffsetY](#) (void) const  
Return the Y rendering offset.
- const [String](#) & `getName` (void) const  
Return the name of this [Image](#) object.
- const [String](#) & `getImagesetName` (void) const  
Return the name of the [Imageset](#) that contains this [Image](#).
- const [Imageset](#) \* `getImageset` (void) const  
Return the parent [Imageset](#) object that contains this [Image](#).
- const [Rect](#) & `getSourceTextureArea` (void) const  
Return [Rect](#) describing the source texture area used by this [Image](#).



- void `draw` (const `Vector3` &position, const `Size` &size, const `Rect` &clip\_rect, const `colour` &top\_left\_colour=0xFFFFFFFF, const `colour` &top\_right\_colour=0xFFFFFFFF, const `colour` &bottom\_left\_colour=0xFFFFFFFF, const `colour` &bottom\_right\_colour=0xFFFFFFFF, `QuadSplitMode` quad\_split\_mode=TopLeftToBottomRight) const

*Queue the image to be drawn.*

- void `draw` (const `Rect` &dest\_rect, float z, const `Rect` &clip\_rect, const `colour` &top\_left\_colour=0xFFFFFFFF, const `colour` &top\_right\_colour=0xFFFFFFFF, const `colour` &bottom\_left\_colour=0xFFFFFFFF, const `colour` &bottom\_right\_colour=0xFFFFFFFF, `QuadSplitMode` quad\_split\_mode=TopLeftToBottomRight) const

*Queue the image to be drawn.*

- void `draw` (const `Vector3` &position, const `Size` &size, const `Rect` &clip\_rect, const `ColourRect` &colours, `QuadSplitMode` quad\_split\_mode=TopLeftToBottomRight) const

*Queue the image to be drawn.*

- void `draw` (const `Vector3` &position, const `Rect` &clip\_rect, const `ColourRect` &colours, `QuadSplitMode` quad\_split\_mode=TopLeftToBottomRight) const

*Queue the image to be drawn.*

- void `draw` (const `Vector3` &position, const `Rect` &clip\_rect, const `colour` &top\_left\_colour=0xFFFFFFFF, const `colour` &top\_right\_colour=0xFFFFFFFF, const `colour` &bottom\_left\_colour=0xFFFFFFFF, const `colour` &bottom\_right\_colour=0xFFFFFFFF, `QuadSplitMode` quad\_split\_mode=TopLeftToBottomRight) const

*Queue the image to be drawn.*

- void `draw` (const `Rect` &dest\_rect, float z, const `Rect` &clip\_rect, const `ColourRect` &colours, `QuadSplitMode` quad\_split\_mode=TopLeftToBottomRight) const

*Queue the image to be drawn.*

- void `writeXMLToStream` (`XMLSerializer` &xml\_stream) const

*Writes an xml representation of this `Image` object to out\_stream.*

- `Image` (void)

*Default constructor (only used by std::map).*

- `Image` (const `Imageset` \*owner, const `String` &name, const `Rect` &area, const `Point` &render\_offset, float horzScaling=1.0f, float vertScaling=1.0f)

*Constructor for `Image` objects. This is not normally used directly by client code, use the `Imageset` interface instead.*

- `Image` (const `Image` &image)

*Copy constructor.*

- `~Image` (void)

*Destructor for `Image` objects.*

## Friends

- class `std::map< String, Image, String::FastLessCompare >`
- struct `std::pair< const String, Image >`
- class `Imageset`

### 6.125.1 Detailed Description

Class that represents a single [Image](#) of an [Imageset](#).

### 6.125.2 Constructor & Destructor Documentation

#### 6.125.2.1 CEGUI::Image::Image (const Imageset \* owner, const String & name, const Rect & area, const Point & render\_offset, float horzScaling = 1.0f, float vertScaling = 1.0f)

Constructor for [Image](#) objects. This is not normally used directly by client code, use the [Imageset](#) interface instead.

#### Parameters:

- owner* Pointer to a [Imageset](#) object that owns this [Image](#). This must not be NULL.
- name* [String](#) object describing the name of the image being created.
- area* [Rect](#) object describing an area that will be associated with this image.
- render\_offset* [Point](#) object that describes the offset to be applied when rendering this image.
- horzScaling* float value indicating the initial horizontal scaling to be applied to this image.
- vertScaling* float value indicating the initial vertical scaling to be applied to this image.

#### Exceptions:

- [NullObjectException](#) Thrown if *owner* was NULL.

### 6.125.3 Member Function Documentation

#### 6.125.3.1 Size CEGUI::Image::getSize (void) const [inline]

Return a [Size](#) object containing the dimensions of the [Image](#).

#### Returns:

- [Size](#) object holding the width and height of the [Image](#).

#### 6.125.3.2 float CEGUI::Image::getWidth (void) const [inline]

Return the pixel width of the image.

#### Returns:

- Width of this [Image](#) in pixels.

**6.125.3.3 float CEGUI::Image::getHeight (void) const** [inline]

Return the pixel height of the image.

**Returns:**

Height of this [Image](#) in pixels

**6.125.3.4 Point CEGUI::Image::getOffsets (void) const** [inline]

Return a Point object that contains the offset applied when rendering this [Image](#).

**Returns:**

Point object containing the offsets applied when rendering this [Image](#)

**6.125.3.5 float CEGUI::Image::getOffsetX (void) const** [inline]

Return the X rendering offset.

**Returns:**

X rendering offset. This is the number of pixels that the image is offset by when rendering at any given location.

**6.125.3.6 float CEGUI::Image::getOffsetY (void) const** [inline]

Return the Y rendering offset.

**Returns:**

Y rendering offset. This is the number of pixels that the image is offset by when rendering at any given location.

**6.125.3.7 const String & CEGUI::Image::getName (void) const**

Return the name of this [Image](#) object.

**Returns:**

[String](#) object containing the name of this [Image](#)

**6.125.3.8 const String & CEGUI::Image::getImagesetName (void) const**

Return the name of the [ImageSet](#) that contains this [Image](#).

**Returns:**

[String](#) object containing the name of the [ImageSet](#) which this [Image](#) is a part of.

**6.125.3.9** `const Imageset* CEGUI::Image::getImageset (void) const` [inline]

Return the parent [Imageset](#) object that contains this [Image](#).

**Returns:**

The parent [Imageset](#) object.

**6.125.3.10** `const Rect & CEGUI::Image::getSourceTextureArea (void) const`

Return [Rect](#) describing the source texture area used by this [Image](#).

**Returns:**

[Rect](#) object that describes, in pixels, the area upon the source texture which is used when rendering this [Image](#).

**6.125.3.11** `void CEGUI::Image::draw (const Vector3 & position, const Size & size, const Rect & clip_rect, const colour & top_left_colour = 0xFFFFFFFF, const colour & top_right_colour = 0xFFFFFFFF, const colour & bottom_left_colour = 0xFFFFFFFF, const colour & bottom_right_colour = 0xFFFFFFFF, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const` [inline]

Queue the image to be drawn.

**Note:**

The final position of the [Image](#) will be adjusted by the offset values defined for this [Image](#) object. If absolute positioning is essential then these values should be taken into account prior to calling the [draw\(\)](#) methods. However, by doing this you take away the ability of the [Imageset](#) designer to adjust the alignment and positioning of Images, therefore your component is far less useful since it requires code changes to modify image positioning that could have been handled from a data file.

**Parameters:**

*position* [Vector3](#) object containing the location where the [Image](#) is to be drawn

*size* [Size](#) object describing the size that the [Image](#) is to be drawn at.

*clip\_rect* [Rect](#) object that defines an on-screen area that the [Image](#) will be clipped to when drawing.

*top\_left\_colour* Colour (as 0xAARRGGBB value) to be applied to the top-left corner of the [Image](#).

*top\_right\_colour* Colour (as 0xAARRGGBB value) to be applied to the top-right corner of the [Image](#).

*bottom\_left\_colour* Colour (as 0xAARRGGBB value) to be applied to the bottom-left corner of the [Image](#).

*bottom\_right\_colour* Colour (as 0xAARRGGBB value) to be applied to the bottom-right corner of the [Image](#).

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

**Returns:**

Nothing

**6.125.3.12** `void CEGUI::Image::draw (const Rect & dest_rect, float z, const Rect & clip_rect, const colour & top_left_colour = 0xFFFFFFFF, const colour & top_right_colour = 0xFFFFFFFF, const colour & bottom_left_colour = 0xFFFFFFFF, const colour & bottom_right_colour = 0xFFFFFFFF, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const` [inline]

Queue the image to be drawn.

**Note:**

The final position of the [Image](#) will be adjusted by the offset values defined for this [Image](#) object. If absolute positioning is essential then these values should be taken into account prior to calling the [draw\(\)](#) methods. However, by doing this you take away the ability of the [Imageset](#) designer to adjust the alignment and positioning of Images, therefore your component is far less useful since it requires code changes to modify image positioning that could have been handled from a data file.

**Parameters:**

*dest\_rect* [Rect](#) object defining the area on-screen where the [Image](#) is to be drawn. The [Image](#) will be scaled to fit the area as required.

*z* Z-order position for the image. Positions increase "into the screen", so 0.0f is at the top of the z-order.

*clip\_rect* [Rect](#) object that defines an on-screen area that the [Image](#) will be clipped to when drawing.

*top\_left\_colour* Colour (as 0xAARRGGBB value) to be applied to the top-left corner of the [Image](#).

*top\_right\_colour* Colour (as 0xAARRGGBB value) to be applied to the top-right corner of the [Image](#).

*bottom\_left\_colour* Colour (as 0xAARRGGBB value) to be applied to the bottom-left corner of the [Image](#).

*bottom\_right\_colour* Colour (as 0xAARRGGBB value) to be applied to the bottom-right corner of the [Image](#).

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

**Returns:**

Nothing

**6.125.3.13** `void CEGUI::Image::draw (const Vector3 & position, const Size & size, const Rect & clip_rect, const ColourRect & colours, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const` [inline]

Queue the image to be drawn.

**Note:**

The final position of the [Image](#) will be adjusted by the offset values defined for this [Image](#) object. If absolute positioning is essential then these values should be taken into account prior to calling the [draw\(\)](#) methods. However, by doing this you take away the ability of the [Imageset](#) designer to adjust the alignment and positioning of Images, therefore your component is far less useful since it requires code changes to modify image positioning that could have been handled from a data file.

**Parameters:**

*position* [Vector3](#) object containing the location where the [Image](#) is to be drawn

*size* [Size](#) object describing the size that the [Image](#) is to be drawn at.

*clip\_rect* [Rect](#) object that defines an on-screen area that the [Image](#) will be clipped to when drawing.

*colours* [ColourRect](#) object that describes the [colour](#) values to use for each corner of the [Image](#).

*quad\_split\_mode* One of the [QuadSplitMode](#) values specifying the way quads are split into triangles

#### Returns:

Nothing

**6.125.3.14** `void CEGUI::Image::draw (const Vector3 & position, const Rect & clip_rect, const ColourRect & colours, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const [inline]`

Queue the image to be drawn.

#### Note:

The final position of the [Image](#) will be adjusted by the offset values defined for this [Image](#) object. If absolute positioning is essential then these values should be taken into account prior to calling the [draw\(\)](#) methods. However, by doing this you take away the ability of the [Imageset](#) designer to adjust the alignment and positioning of Images, therefore your component is far less useful since it requires code changes to modify image positioning that could have been handled from a data file.

#### Parameters:

*position* [Vector3](#) object containing the location where the [Image](#) is to be drawn

#### Note:

The image will be drawn at it's internally defined size.

#### Parameters:

*clip\_rect* [Rect](#) object that defines an on-screen area that the [Image](#) will be clipped to when drawing.

*colours* [ColourRect](#) object that describes the [colour](#) values to use for each corner of the [Image](#).

*quad\_split\_mode* One of the [QuadSplitMode](#) values specifying the way quads are split into triangles

#### Returns:

Nothing

**6.125.3.15** `void CEGUI::Image::draw (const Vector3 & position, const Rect & clip_rect, const colour & top_left_colour = 0xFFFFFFFF, const colour & top_right_colour = 0xFFFFFFFF, const colour & bottom_left_colour = 0xFFFFFFFF, const colour & bottom_right_colour = 0xFFFFFFFF, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const [inline]`

Queue the image to be drawn.

#### Note:

The final position of the [Image](#) will be adjusted by the offset values defined for this [Image](#) object. If absolute positioning is essential then these values should be taken into account prior to calling the

[draw\(\)](#) methods. However, by doing this you take away the ability of the [Imageset](#) designer to adjust the alignment and positioning of Images, therefore your component is far less useful since it requires code changes to modify image positioning that could have been handled from a data file.

**Parameters:**

*position* [Vector3](#) object containing the location where the [Image](#) is to be drawn

*clip\_rect* [Rect](#) object that defines an on-screen area that the [Image](#) will be clipped to when drawing.

*top\_left\_colour* Colour (as 0xAARRGGBB value) to be applied to the top-left corner of the [Image](#).

*top\_right\_colour* Colour (as 0xAARRGGBB value) to be applied to the top-right corner of the [Image](#).

*bottom\_left\_colour* Colour (as 0xAARRGGBB value) to be applied to the bottom-left corner of the [Image](#).

*bottom\_right\_colour* Colour (as 0xAARRGGBB value) to be applied to the bottom-right corner of the [Image](#).

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

**Returns:**

Nothing

**6.125.3.16** `void CEGUI::Image::draw (const Rect & dest_rect, float z, const Rect & clip_rect, const ColourRect & colours, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const`

Queue the image to be drawn.

**Note:**

The final position of the [Image](#) will be adjusted by the offset values defined for this [Image](#) object. If absolute positioning is essential then these values should be taken into account prior to calling the [draw\(\)](#) methods. However, by doing this you take away the ability of the [Imageset](#) designer to adjust the alignment and positioning of Images, therefore your component is far less useful since it requires code changes to modify image positioning that could have been handled from a data file.

**Parameters:**

*dest\_rect* [Rect](#) object defining the area on-screen where the [Image](#) is to be drawn. The [Image](#) will be scaled to fit the area as required.

*z* Z-order position for the image. Positions increase "into the screen", so 0.0f is at the top of the z-order.

*clip\_rect* [Rect](#) object that defines an on-screen area that the [Image](#) will be clipped to when drawing.

*colours* [ColourRect](#) object that describes the [colour](#) values to use for each corner of the [Image](#).

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

**Returns:**

Nothing

**6.125.3.17 void CEGUI::Image::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [Image](#) object to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

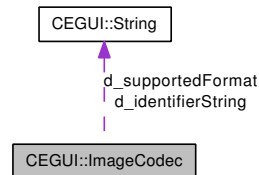
Nothing.



## 6.126 CEGUI::ImageCodec Class Reference

Abstract ImageLoader class. An image loader encapsulate the loading of a texture.

Collaboration diagram for CEGUI::ImageCodec:



### Public Member Functions

- virtual `~ImageCodec()`  
*Destructor.*
- const `String` & `getIdentifierString()` const  
*Return the name of the image codec object.*
- const `String` & `getSupportedFormat()` const  
*Return the list of image file format supported.*
- virtual `Texture * load(const RawDataContainer &data, Texture *result)=0`  
*Load an image from a memory buffer.*

### Protected Member Functions

- `ImageCodec(const String &name)`

### Protected Attributes

- `String d_supportedFormat`  
*list all image file format supported*

#### 6.126.1 Detailed Description

Abstract ImageLoader class. An image loader encapsulate the loading of a texture.

This class define the loading of an abstract

#### 6.126.2 Member Function Documentation

##### 6.126.2.1 const String & CEGUI::ImageCodec::getIdentifierString() const

Return the name of the image codec object.

Return the name of the image codec

**Returns:**

a string containing image codec name

**6.126.2.2    `const String & CEGUI::ImageCodec::getSupportedFormat () const`**

Return the list of image file format supported.

Return a list of space separated image format supported by this codec

**Returns:**

list of supported image file format separated with space

**6.126.2.3    `virtual Texture* CEGUI::ImageCodec::load (const RawDataContainer & data, Texture * result)    [pure virtual]`**

Load an image from a memory buffer.

**Parameters:**

*data*    the image data

*result*    the texture to use for storing the image data

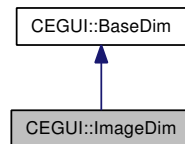
**Returns:**

result on success or 0 if the load failed

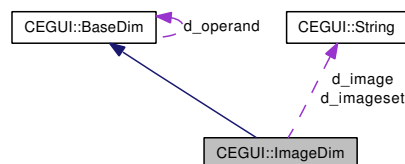
## 6.127 CEGUI::ImageDim Class Reference

**Dimension** type that represents some dimension of a named **Image**. Implements **BaseDim** interface.

Inheritance diagram for CEGUI::ImageDim:



Collaboration diagram for CEGUI::ImageDim:



### Public Member Functions

- **ImageDim** (const **String** &imageset, const **String** &image, **DimensionType** dim)  
*Constructor.*
- void **setSourceImage** (const **String** &imageset, const **String** &image)  
*Sets the source image information for this **ImageDim**.*
- void **setSourceDimension** (**DimensionType** dim)  
*Sets the source dimension type for this **ImageDim**.*

### Protected Member Functions

- float **getValue\_impl** (const **Window** &wnd) const  
*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically.*
- float **getValue\_impl** (const **Window** &wnd, const **Rect** &container) const  
*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically by **BaseDim**.*
- void **writeXMLElementName\_impl** (**XMLSerializer** &xml\_stream) const  
*Implementataion method to output real xml element name.*
- void **writeXMLElementAttributes\_impl** (**XMLSerializer** &xml\_stream) const  
*Implementataion method to create the element attributes.*

- [BaseDim](#) \* [clone\\_impl](#) () const

*Implementataion method to return a clone of this sub-class of [BaseDim](#). This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by [BaseDim](#).*

### 6.127.1 Detailed Description

[Dimension](#) type that represents some dimension of a named [Image](#). Implements [BaseDim](#) interface.

### 6.127.2 Constructor & Destructor Documentation

#### 6.127.2.1 CEGUI::ImageDim::ImageDim (const String & *imageset*, const String & *image*, DimensionType *dim*)

Constructor.

##### Parameters:

*imageset* [String](#) object holding the name of the imageset which contains the image.

*image* [String](#) object holding the name of the image.

*dim* DimensionType value indicating which dimension of the described image that this [ImageDim](#) is to represent.

### 6.127.3 Member Function Documentation

#### 6.127.3.1 void CEGUI::ImageDim::setSourceImage (const String & *imageset*, const String & *image*)

Sets the source image information for this [ImageDim](#).

##### Parameters:

*imageset* [String](#) object holding the name of the imageset which contains the image.

*image* [String](#) object holding the name of the image.

##### Returns:

Nothing.

#### 6.127.3.2 void CEGUI::ImageDim::setSourceDimension (DimensionType *dim*)

Sets the source dimension type for this [ImageDim](#).

##### Parameters:

*dim* DimensionType value indicating which dimension of the described image that this [ImageDim](#) is to represent.

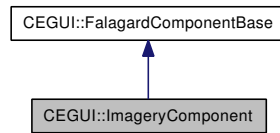
##### Returns:

Nothing.

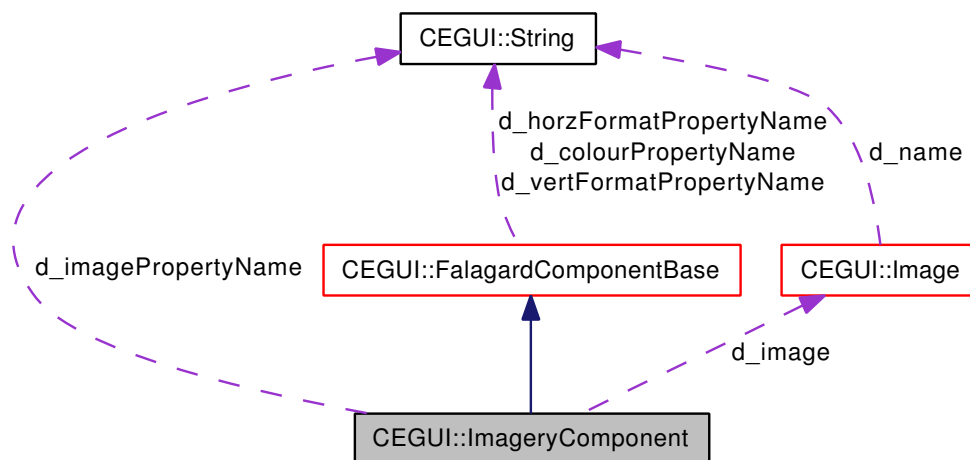
## 6.128 CEGUI::ImageryComponent Class Reference

Class that encapsulates information for a single image component.

Inheritance diagram for CEGUI::ImageryComponent:



Collaboration diagram for CEGUI::ImageryComponent:



### Public Member Functions

- **ImageryComponent** ()  
*Constructor.*
- const **Image** \* **getImage** () const  
*Return the **Image** object that will be drawn by this **ImageryComponent**.*
- void **setImage** (const **Image** \*image)  
*Set the **Image** that will be drawn by this **ImageryComponent**.*
- void **setImage** (const **String** &imageset, const **String** &image)  
*Set the **Image** that will be drawn by this **ImageryComponent**.*
- **VerticalFormatting** **getVerticalFormatting** () const  
*Return the current vertical formatting setting for this **ImageryComponent**.*
- void **setVerticalFormatting** (**VerticalFormatting** fmt)  
*Set the vertical formatting setting for this **ImageryComponent**.*

- [HorizontalFormatting](#) [getHorizontalFormatting](#) () const  
*Return the current horizontal formatting setting for this [ImageryComponent](#).*
- void [setHorizontalFormatting](#) ([HorizontalFormatting](#) fmt)  
*Set the horizontal formatting setting for this [ImageryComponent](#).*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [ImageryComponent](#) to out\_stream.*
- bool [isImageFetchedFromProperty](#) () const  
*Return whether this [ImageryComponent](#) fetches it's image via a property on the target window.*
- const [String](#) & [getImagePropertySource](#) () const  
*Return the name of the property that will be used to determine the image for this [ImageryComponent](#).*
- void [setImagePropertySource](#) (const [String](#) &property)  
*Set the name of the property that will be used to determine the image for this [ImageryComponent](#).*

## Protected Member Functions

- void [render\\_impl](#) ([Window](#) &srcWindow, [Rect](#) &destRect, float base\_z, const [CEGUI::ColourRect](#) \*modColours, const [Rect](#) \*clipper, bool clipToDisplay) const  
*Method to do main render caching work.*

## Protected Attributes

- const [Image](#) \* [d\\_image](#)  
*[CEGUI::Image](#) to be drawn by this image component.*
- [VerticalFormatting](#) [d\\_vertFormatting](#)  
*Vertical formatting to be applied when rendering the image component.*
- [HorizontalFormatting](#) [d\\_horzFormatting](#)  
*Horizontal formatting to be applied when rendering the image component.*
- [String](#) [d\\_imagePropertyName](#)  
*Name of the property to access to obtain the image to be used.*

### 6.128.1 Detailed Description

Class that encapsulates information for a single image component.

## 6.128.2 Member Function Documentation

### 6.128.2.1 `const Image * CEGUI::ImageryComponent::getImage (void) const`

Return the [Image](#) object that will be drawn by this [ImageryComponent](#).

**Returns:**

[Image](#) object.

### 6.128.2.2 `void CEGUI::ImageryComponent::setImage (const Image * image)`

Set the [Image](#) that will be drawn by this [ImageryComponent](#).

**Parameters:**

*Pointer* to the [Image](#) object to be drawn by this [ImageryComponent](#).

**Returns:**

Nothing.

### 6.128.2.3 `void CEGUI::ImageryComponent::setImage (const String & imageset, const String & image)`

Set the [Image](#) that will be drawn by this [ImageryComponent](#).

**Parameters:**

*imageset* [String](#) holding the name of the Imagset that contains the [Image](#) to be rendered.

*image* [String](#) holding the name of the [Image](#) to be rendered.

**Returns:**

Nothing.

### 6.128.2.4 `VerticalFormatting CEGUI::ImageryComponent::getVerticalFormatting () const`

Return the current vertical formatting setting for this [ImageryComponent](#).

**Returns:**

One of the VerticalFormatting enumerated values.

### 6.128.2.5 `void CEGUI::ImageryComponent::setVerticalFormatting (VerticalFormatting fnt)`

Set the vertical formatting setting for this [ImageryComponent](#).

**Parameters:**

*fnt* One of the VerticalFormatting enumerated values.

**Returns:**

Nothing.

**6.128.2.6 HorizontalFormatting CEGUI::ImageryComponent::getHorizontalFormatting () const**

Return the current horizontal formatting setting for this [ImageryComponent](#).

**Returns:**

One of the HorizontalFormatting enumerated values.

**6.128.2.7 void CEGUI::ImageryComponent::setHorizontalFormatting (HorizontalFormatting *fmt*)**

Set the horizontal formatting setting for this [ImageryComponent](#).

**Parameters:**

*fmt* One of the HorizontalFormatting enumerated values.

**Returns:**

Nothing.

**6.128.2.8 void CEGUI::ImageryComponent::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [ImageryComponent](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.128.2.9 bool CEGUI::ImageryComponent::isImageFetchedFromProperty () const**

Return whether this [ImageryComponent](#) fetches it's image via a property on the target window.

**Returns:**

- true if the image comes via a Property.
- false if the image is defined explicitly.



**6.128.2.10** `const String & CEGUI::ImageryComponent::getImagePropertySource () const`

Return the name of the property that will be used to determine the image for this [ImageryComponent](#).

**Returns:**

[String](#) object holding the name of a Property.

**6.128.2.11** `void CEGUI::ImageryComponent::setImagePropertySource (const String & property)`

Set the name of the property that will be used to determine the image for this [ImageryComponent](#).

**Parameters:**

*property* [String](#) object holding the name of a Property. The property should access a imageset & image specification.

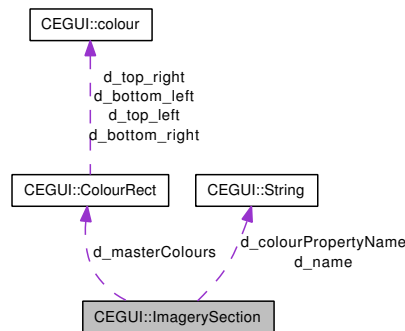
**Returns:**

Nothing.

## 6.129 CEGUI::ImagerySection Class Reference

Class that encapsulates a re-usable collection of imagery specifications.

Collaboration diagram for CEGUI::ImagerySection:



### Public Member Functions

- [ImagerySection](#) ()  
*Constructor.*
- [ImagerySection](#) (const [String](#) &name)  
*ImagerySection* constructor. Name must be supplied, masterColours are set to 0xFFFFFFFF by default.
- void [render](#) ([Window](#) &srcWindow, float base\_z, const [CEGUI::ColourRect](#) \*modColours=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const  
*Render the ImagerySection.*
- void [render](#) ([Window](#) &srcWindow, const [Rect](#) &baseRect, float base\_z, const [CEGUI::ColourRect](#) \*modColours=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const  
*Render the ImagerySection.*
- void [addImageryComponent](#) (const [ImageryComponent](#) &img)  
*Add an ImageryComponent to this ImagerySection.*
- void [clearImageryComponents](#) ()  
*Clear all ImageryComponents from this ImagerySection.*
- void [addTextComponent](#) (const [TextComponent](#) &text)  
*Add a TextComponent to this ImagerySection.*
- void [clearTextComponents](#) ()  
*Clear all TextComponents from this ImagerySection.*
- void [clearFrameComponents](#) ()  
*Clear all FrameComponents from this ImagerySection.*
- void [addFrameComponent](#) (const [FrameComponent](#) &frame)

Add a *FrameComponent* to this *ImagerySection*.

- const *ColourRect* & *getMasterColours* () const  
Return the current master colours set for this *ImagerySection*.
- void *setMasterColours* (const *ColourRect* &cols)  
Set the master colours to be used for this *ImagerySection*.
- const *String* & *getName* () const  
Return the name of this *ImagerySection*.
- void *setMasterColoursPropertySource* (const *String* &property)  
Set the name of the property where master *colour* values can be obtained.
- void *setMasterColoursPropertyIsColourRect* (bool setting=true)  
Set whether the master colours property source represents a full *ColourRect*.
- *Rect* *getBoundingRect* (const *Window* &wnd) const  
Return smallest *Rect* that could contain all imagery within this section.
- *Rect* *getBoundingRect* (const *Window* &wnd, const *Rect* &rect) const  
Return smallest *Rect* that could contain all imagery within this section.
- void *writeXMLToStream* (*XMLSerializer* &xml\_stream) const  
Writes an xml representation of this *ImagerySection* to out\_stream.

## Protected Member Functions

- void *initMasterColourRect* (const *Window* &wnd, *ColourRect* &cr) const  
Helper method to initialise a *ColourRect* with appropriate values according to the way the *ImagerySection* is set up.

### 6.129.1 Detailed Description

Class that encapsulates a re-usable collection of imagery specifications.

### 6.129.2 Constructor & Destructor Documentation

#### 6.129.2.1 CEGUI::ImagerySection::ImagerySection (const *String* & *name*)

*ImagerySection* constructor. Name must be supplied, masterColours are set to 0xFFFFFFFF by default.

#### Parameters:

*name* Name of the new *ImagerySection*.

### 6.129.3 Member Function Documentation

**6.129.3.1** void CEGUI::ImagerySection::render (Window & *srcWindow*, float *base\_z*, const CEGUI::ColourRect \* *modColours* = 0, const Rect \* *clipper* = 0, bool *clipToDisplay* = false) const

Render the [ImagerySection](#).

**Parameters:**

*srcWindow* [Window](#) object to be used when calculating pixel values from [BaseDim](#) values.

*base\_z* base z value to be used for all imagery in the section.

*modColours* [ColourRect](#) specifying colours to be modulated with the ImagerySection's master colours. May be 0.

**Returns:**

Nothing.

**6.129.3.2** void CEGUI::ImagerySection::render (Window & *srcWindow*, const Rect & *baseRect*, float *base\_z*, const CEGUI::ColourRect \* *modColours* = 0, const Rect \* *clipper* = 0, bool *clipToDisplay* = false) const

Render the [ImagerySection](#).

**Parameters:**

*srcWindow* [Window](#) object to be used when calculating pixel values from [BaseDim](#) values.

*baseRect* [Rect](#) object to be used when calculating pixel values from [BaseDim](#) values.

*base\_z* base z value to be used for all imagery in the section.

*modColours* [ColourRect](#) specifying colours to be modulated with the ImagerySection's master colours. May be 0.

**Returns:**

Nothing.

**6.129.3.3** void CEGUI::ImagerySection::addImageryComponent (const ImageryComponent & *img*)

Add an [ImageryComponent](#) to this [ImagerySection](#).

**Parameters:**

*img* [ImageryComponent](#) to be added to the section (a copy is made)

**Returns:**

Nothing

**6.129.3.4 void CEGUI::ImagerySection::clearImageryComponents ()**

Clear all ImageryComponents from this [ImagerySection](#).

**Returns:**

Nothing

**6.129.3.5 void CEGUI::ImagerySection::addTextComponent (const TextComponent & *text*)**

Add a [TextComponent](#) to this [ImagerySection](#).

**Parameters:**

*text* [TextComponent](#) to be added to the section (a copy is made)

**Returns:**

Nothing

**6.129.3.6 void CEGUI::ImagerySection::clearTextComponents ()**

Clear all TextComponents from this [ImagerySection](#).

**Returns:**

Nothing

**6.129.3.7 void CEGUI::ImagerySection::clearFrameComponents ()**

Clear all FrameComponents from this [ImagerySection](#).

**Returns:**

Nothing

**6.129.3.8 void CEGUI::ImagerySection::addFrameComponent (const FrameComponent & *frame*)**

Add a [FrameComponent](#) to this [ImagerySection](#).

**Parameters:**

*frame* [FrameComponent](#) to be added to the section (a copy is made)

**Returns:**

Nothing

**6.129.3.9 const ColourRect & CEGUI::ImagerySection::getMasterColours () const**

Return the current master colours set for this [ImagerySection](#).

**Returns:**

[ColourRect](#) describing the master [colour](#) values in use for this [ImagerySection](#).

**6.129.3.10 void CEGUI::ImagerySection::setMasterColours (const ColourRect & cols)**

Set the master colours to be used for this [ImagerySection](#).

**Parameters:**

*cols* [ColourRect](#) describing the colours to be set as the master colours for this [ImagerySection](#).

**Returns:**

Nothing.

**6.129.3.11 const String & CEGUI::ImagerySection::getName () const**

Return the name of this [ImagerySection](#).

**Returns:**

[String](#) object holding the name of the [ImagerySection](#).

**6.129.3.12 void CEGUI::ImagerySection::setMasterColoursPropertySource (const String & property)**

Set the name of the property where master [colour](#) values can be obtained.

**Parameters:**

*property* [String](#) containing the name of the property.

**Returns:**

Nothing.

**6.129.3.13 void CEGUI::ImagerySection::setMasterColoursPropertyIsColourRect (bool setting = true)**

Set whether the master colours property source represents a full [ColourRect](#).

**Parameters:**

- setting* • true if the master colours property will access a [ColourRect](#) object.
- false if the master colours property will access a [colour](#) object.

**Returns:**

Nothing.

**6.129.3.14 void CEGUI::ImagerySection::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [ImagerySection](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.129.3.15 void CEGUI::ImagerySection::initMasterColourRect (const Window & *wnd*, ColourRect & *cr*) const** [protected]

Helper method to initialise a [ColourRect](#) with appropriate values according to the way the [ImagerySection](#) is set up.

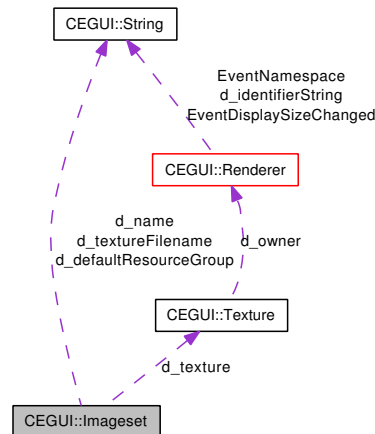
This will try and get values from multiple places:

- a property attached to *wnd*
- or the integral `d_masterColours` value.

## 6.130 CEGUI::Imageset Class Reference

Offers functions to define, access, and draw, a set of image components on a single graphical surface or [Texture](#).

Collaboration diagram for CEGUI::Imageset:



### Public Types

- typedef [ConstBaseIterator](#)< ImageRegistry > [ImageIterator](#)  
*Iterator type for this collection.*

### Public Member Functions

- [~Imageset](#) (void)  
*Destroys [Imageset](#) objects.*
- [Texture](#) \* [getTexture](#) (void) const  
*return [Texture](#) object for this [Imageset](#)*
- const [String](#) & [getName](#) (void) const  
*return [String](#) object holding the name of the [Imageset](#)*
- uint [getImageCount](#) (void) const  
*return number of images defined for this [Imageset](#)*
- bool [isImageDefined](#) (const [String](#) &name) const  
*return true if an [Image](#) with the specified name exists.*
- const [Image](#) & [getImage](#) (const [String](#) &name) const  
*return a copy of the [Image](#) object for the named image*
- void [undefineImage](#) (const [String](#) &name)  
*remove the definition for the [Image](#) with the specified name. If no such [Image](#) exists, nothing happens.*



- void `undefineAllImages` (void)  
*Removes the definitions for all [Image](#) objects currently defined in the [ImageSet](#).*
- [Size](#) `getImageSize` (const [String](#) &name) const  
*return a [Size](#) object describing the dimensions of the named image.*
- float `getImageWidth` (const [String](#) &name) const  
*return the width of the named image.*
- float `getImageHeight` (const [String](#) &name) const  
*return the height of the named image.*
- [Point](#) `getImageOffset` (const [String](#) &name) const  
*return the rendering offsets applied to the named image.*
- float `getImageOffsetX` (const [String](#) &name) const  
*return the x rendering offset for the named image.*
- float `getImageOffsetY` (const [String](#) &name) const  
*return the y rendering offset for the named image.*
- void `defineImage` (const [String](#) &name, const [Point](#) &position, const [Size](#) &size, const [Point](#) &render\_offset)  
*Define a new [Image](#) for this [ImageSet](#).*
- void `defineImage` (const [String](#) &name, const [Rect](#) &image\_rect, const [Point](#) &render\_offset)  
*Define a new [Image](#) for this [ImageSet](#).*
- void `draw` (const [Rect](#) &source\_rect, const [Rect](#) &dest\_rect, float z, const [Rect](#) &clip\_rect, const [ColourRect](#) &colours, [QuadSplitMode](#) quad\_split\_mode) const  
*Queues an area of the associated [Texture](#) to be drawn on the screen. Low-level routine to be used carefully!*
- void `draw` (const [Rect](#) &source\_rect, const [Rect](#) &dest\_rect, float z, const [Rect](#) &clip\_rect, const [colour](#) &top\_left\_colour=0xFFFFFFFF, const [colour](#) &top\_right\_colour=0xFFFFFFFF, const [colour](#) &bottom\_left\_colour=0xFFFFFFFF, const [colour](#) &bottom\_right\_colour=0xFFFFFFFF, [QuadSplitMode](#) quad\_split\_mode=TopLeftToBottomRight) const  
*Queues an area of the associated [Texture](#) to be drawn on the screen. Low-level routine to be used carefully!*
- bool `isAutoScaled` (void) const  
*Return whether this [ImageSet](#) is auto-scaled.*
- [Size](#) `getNativeResolution` (void) const  
*Return the native display size for this [ImageSet](#). This is only relevant if the [ImageSet](#) is being auto-scaled.*
- void `setAutoScalingEnabled` (bool setting)  
*Enable or disable auto-scaling for this [ImageSet](#).*
- void `setNativeResolution` (const [Size](#) &size)  
*Set the native resolution for this [ImageSet](#).*

- void [notifyScreenResolution](#) (const [Size](#) &size)  
*Notify the [Imageset](#) of the current (usually new) display resolution.*
- [ImageIterator](#) [getImageIterator](#) (void) const  
*Return an [Imageset::ImageIterator](#) object that can be used to iterate over the [Image](#) objects in the [Imageset](#).*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [Imageset](#) to out\_stream.*

## Static Public Member Functions

- static void [setDefaultResourceGroup](#) (const [String](#) &resourceGroup)  
*Sets the default resource group to be used when loading imageset data.*
- static const [String](#) & [getDefaultResourceGroup](#) ()  
*Returns the default resource group currently set for Imagesets.*

## Protected Member Functions

- void [load](#) (const [String](#) &filename, const [String](#) &resourceGroup)  
*Initialise the [Imageset](#) with information taken from the specified file.*
- void [unload](#) (void)  
*Unloads all loaded data and leaves the [Imageset](#) in a clean (but un-usable) state. This should be called for cleanup purposes only.*
- void [setTexture](#) ([Texture](#) \*texture)  
*set the [Texture](#) object to be used by this [Imageset](#). Changing textures on an [Imageset](#) that is in use is not a good idea!*
- void [updateImageScalingFactors](#) (void)  
*Sets the scaling factor for all Images that are a part of this [Imageset](#).*

## Protected Attributes

- [String](#) [d\\_name](#)  
*Holds the name of this imageset.*
- [ImageRegistry](#) [d\\_images](#)  
*Registry of [Image](#) objects for the images defined for this [Imageset](#).*
- [Texture](#) \* [d\\_texture](#)  
*[Texture](#) object that handles imagery for this [Imageset](#).*
- [String](#) [d\\_textureFilename](#)

*String* holding the name of the texture filename (if any).

- bool `d_autoScale`  
*true when auto-scaling is enabled.*
- float `d_horzScaling`  
*current horizontal scaling factor.*
- float `d_vertScaling`  
*current vertical scaling factor.*
- float `d_nativeHorzRes`  
*native horizontal resolution for this *Imageset*.*
- float `d_nativeVertRes`  
*native vertical resolution for this *Imageset*.*

## Static Protected Attributes

- static const char `ImagesetSchemaName` [] = "Imageset.xsd"  
*Filename of the XML schema used for validating *Imageset* files.*
- static `String` `d_defaultResourceGroup`  
*Default resource group specifically for *Imagesets*.*

## Friends

- class `Imageset_xmlHandler`
- `Imageset * ImagesetManager::createImageset` (const `String` &name, `Texture *texture`)
- `Imageset * ImagesetManager::createImageset` (const `String` &filename, const `String` &resourceGroup)
- `Imageset * ImagesetManager::createImagesetFromImageFile` (const `String` &name, const `String` &filename, const `String` &resourceGroup)
- void `ImagesetManager::destroyImageset` (const `String` &name)

### 6.130.1 Detailed Description

Offers functions to define, access, and draw, a set of image components on a single graphical surface or `Texture`.

`Imageset` objects are a means by which a single graphical image (file, `Texture`, etc), can be split into a number of 'components' which can later be accessed via name. The components of an `Imageset` can queried for various details, and sent to the `Renderer` object for drawing.

## 6.130.2 Member Function Documentation

### 6.130.2.1 `Texture* CEGUI::Imageset::getTexture (void) const` [inline]

return [Texture](#) object for this [Imageset](#)

**Returns:**

[Texture](#) object that holds the imagery for this [Imageset](#)

### 6.130.2.2 `const String& CEGUI::Imageset::getName (void) const` [inline]

return [String](#) object holding the name of the [Imageset](#)

**Returns:**

[String](#) object that holds the name of the [Imageset](#).

### 6.130.2.3 `uint CEGUI::Imageset::getImageCount (void) const` [inline]

return number of images defined for this [Imageset](#)

**Returns:**

uint value equal to the number of [Image](#) objects defined for the [Imageset](#)

### 6.130.2.4 `bool CEGUI::Imageset::isImageDefined (const String & name) const` [inline]

return true if an [Image](#) with the specified name exists.

**Parameters:**

*name* [String](#) object holding the name of the [Image](#) to look for.

**Returns:**

true if an [Image](#) object named *name* is defined for this [Imageset](#), else false.

### 6.130.2.5 `const Image & CEGUI::Imageset::getImage (const String & name) const`

return a copy of the [Image](#) object for the named image

**Parameters:**

*name* [String](#) object holding the name of the [Image](#) object to be returned

**Returns:**

constant [Image](#) object that has the requested name.

**Exceptions:**

[UnknownObjectException](#) thrown if no [Image](#) named *name* is defined for the [Imageset](#)

**6.130.2.6 void CEGUI::Imageset::undefineImage (const String & name)**

remove the definition for the [Image](#) with the specified name. If no such [Image](#) exists, nothing happens.

**Parameters:**

*name* [String](#) object holding the name of the [Image](#) object to be removed from the [Imageset](#),

**Returns:**

Nothing.

**6.130.2.7 void CEGUI::Imageset::undefineAllImages (void)**

Removes the definitions for all [Image](#) objects currently defined in the [Imageset](#).

**Returns:**

Nothing

**6.130.2.8 Size CEGUI::Imageset::getImageSize (const String & name) const [inline]**

return a [Size](#) object describing the dimensions of the named image.

**Parameters:**

*name* [String](#) object holding the name of the [Image](#).

**Returns:**

[Size](#) object holding the dimensions of the requested [Image](#).

**Exceptions:**

[UnknownObjectException](#) thrown if no [Image](#) named *name* is defined for the [Imageset](#)

**6.130.2.9 float CEGUI::Imageset::getImageWidth (const String & name) const [inline]**

return the width of the named image.

**Parameters:**

*name* [String](#) object holding the name of the [Image](#).

**Returns:**

float value equalling the width of the requested [Image](#).

**Exceptions:**

[UnknownObjectException](#) thrown if no [Image](#) named *name* is defined for the [Imageset](#)

**6.130.2.10** float CEGUI::Imageset::getImageHeight (const String & *name*) const [inline]

return the height of the named image.

**Parameters:**

*name* String object holding the name of the Image.

**Returns:**

float value equalling the height of the requested Image.

**Exceptions:**

*UnknownObjectException* thrown if no Image named *name* is defined for the Imageset

**6.130.2.11** Point CEGUI::Imageset::getImageOffset (const String & *name*) const [inline]

return the rendering offsets applied to the named image.

**Parameters:**

*name* String object holding the name of the Image.

**Returns:**

Point object that holds the rendering offsets for the requested Image.

**Exceptions:**

*UnknownObjectException* thrown if no Image named *name* is defined for the Imageset

**6.130.2.12** float CEGUI::Imageset::getImageOffsetX (const String & *name*) const [inline]

return the x rendering offset for the named image.

**Parameters:**

*name* String object holding the name of the Image.

**Returns:**

float value equal to the x rendering offset applied when drawing the requested Image.

**Exceptions:**

*UnknownObjectException* thrown if no Image named *name* is defined for the Imageset

**6.130.2.13** float CEGUI::Imageset::getImageOffsetY (const String & *name*) const [inline]

return the y rendering offset for the named image.

**Parameters:**

*name* [String](#) object holding the name of the [Image](#).

**Returns:**

float value equal to the y rendering offset applied when drawing the requested [Image](#).

**Exceptions:**

[UnknownObjectException](#) thrown if no [Image](#) named *name* is defined for the [Imageset](#)

**6.130.2.14 void CEGUI::Imageset::defineImage (const String & name, const Point & position, const Size & size, const Point & render\_offset) [inline]**

Define a new [Image](#) for this [Imageset](#).

**Parameters:**

*name* [String](#) object holding the name that will be assigned to the new [Image](#), which must be unique within the [Imageset](#).

*position* [Point](#) object describing the pixel location of the [Image](#) on the image file / texture associated with this [Imageset](#).

*size* [Size](#) object describing the dimensions of the [Image](#), in pixels.

*render\_offset* [Point](#) object describing the offsets, in pixels, that are to be applied to the [Image](#) when it is drawn.

**Returns:**

Nothing

**Exceptions:**

[AlreadyExistsException](#) thrown if an [Image](#) named *name* is already defined for this [Imageset](#)

**6.130.2.15 void CEGUI::Imageset::defineImage (const String & name, const Rect & image\_rect, const Point & render\_offset)**

Define a new [Image](#) for this [Imageset](#).

**Parameters:**

*name* [String](#) object holding the name that will be assigned to the new [Image](#), which must be unique within the [Imageset](#).

*image\_rect* [Rect](#) object describing the area on the image file / texture associated with this [Imageset](#) that will be used for the [Image](#).

*render\_offset* [Point](#) object describing the offsets, in pixels, that are to be applied to the [Image](#) when it is drawn.

**Returns:**

Nothing

**Exceptions:**

[AlreadyExistsException](#) thrown if an [Image](#) named *name* is already defined for this [Imageset](#)

**6.130.2.16** `void CEGUI::Imageset::draw (const Rect & source_rect, const Rect & dest_rect, float z, const Rect & clip_rect, const ColourRect & colours, QuadSplitMode quad_split_mode) const`

Queues an area of the associated [Texture](#) to be drawn on the screen. Low-level routine to be used carefully!

**Parameters:**

*source\_rect* [Rect](#) object describing the area of the image file / texture that is to be queued for drawing

*dest\_rect* [Rect](#) describing the area of the screen that will be filled with the imagery from *source\_rect*.

*z* float value specifying 'z' order. 0 is topmost with increasing values moving back into the screen.

*clip\_rect* [Rect](#) object describing a 'clipping rectangle' that will be applied when drawing the requested imagery

*colours* [ColourRect](#) object holding the ARGB colours to be applied to the four corners of the rendered imagery.

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

**Returns:**

Nothing

**6.130.2.17** `void CEGUI::Imageset::draw (const Rect & source_rect, const Rect & dest_rect, float z, const Rect & clip_rect, const colour & top_left_colour = 0xFFFFFFFF, const colour & top_right_colour = 0xFFFFFFFF, const colour & bottom_left_colour = 0xFFFFFFFF, const colour & bottom_right_colour = 0xFFFFFFFF, QuadSplitMode quad_split_mode = TopLeftToBottomRight) const [inline]`

Queues an area of the associated [Texture](#) to be drawn on the screen. Low-level routine to be used carefully!

**Parameters:**

*source\_rect* [Rect](#) object describing the area of the image file / texture that is to be queued for drawing

*dest\_rect* [Rect](#) describing the area of the screen that will be filled with the imagery from *source\_rect*.

*z* float value specifying 'z' order. 0 is topmost with increasing values moving back into the screen.

*clip\_rect* [Rect](#) object describing a 'clipping rectangle' that will be applied when drawing the requested imagery

*top\_left\_colour* [colour](#) to be applied to the top left corner of the rendered imagery.

*top\_right\_colour* [colour](#) to be applied to the top right corner of the rendered imagery.

*bottom\_left\_colour* [colour](#) to be applied to the bottom left corner of the rendered imagery.

*bottom\_right\_colour* [colour](#) to be applied to the bottom right corner of the rendered imagery.

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

**Returns:**

Nothing



**6.130.2.18** `bool CEGUI::Imageset::isAutoScaled (void) const` `[inline]`

Return whether this [Imageset](#) is auto-scaled.

**Returns:**

true if [Imageset](#) is auto-scaled, false if not.

**6.130.2.19** `Size CEGUI::Imageset::getNativeResolution (void) const` `[inline]`

Return the native display size for this [Imageset](#). This is only relevant if the [Imageset](#) is being auto-scaled.

**Returns:**

[Size](#) object describing the native display size for this [Imageset](#).

**6.130.2.20** `void CEGUI::Imageset::setAutoScalingEnabled (bool setting)`

Enable or disable auto-scaling for this [Imageset](#).

**Parameters:**

*setting* true to enable auto-scaling, false to disable auto-scaling.

**Returns:**

Nothing.

**6.130.2.21** `void CEGUI::Imageset::setNativeResolution (const Size & size)`

Set the native resolution for this [Imageset](#).

**Parameters:**

*size* [Size](#) object describing the new native screen resolution for this [Imageset](#).

**Returns:**

Nothing

**6.130.2.22** `void CEGUI::Imageset::notifyScreenResolution (const Size & size)`

Notify the [Imageset](#) of the current (usually new) display resolution.

**Parameters:**

*size* [Size](#) object describing the display resolution

**Returns:**

Nothing

**6.130.2.23 void CEGUI::Imageset::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [Imageset](#) to *out\_stream*.

**Parameters:**

*out\_stream* Stream where xml data should be output.

*indentLevel* Current XML indentation level

**Returns:**

Nothing.

**6.130.2.24 static void CEGUI::Imageset::setDefaultResourceGroup (const String & *resourceGroup*) [inline, static]**

Sets the default resource group to be used when loading imageset data.

**Parameters:**

*resourceGroup* [String](#) describing the default resource group identifier to be used.

**Returns:**

Nothing.

**6.130.2.25 static const String& CEGUI::Imageset::getDefaultResourceGroup (void) [inline, static]**

Returns the default resource group currently set for Imagesets.

**Returns:**

[String](#) describing the default resource group identifier that will be used when loading [Imageset](#) data.

**6.130.2.26 void CEGUI::Imageset::load (const String & *filename*, const String & *resourceGroup*) [protected]**

Initialise the [Imageset](#) with information taken from the specified file.

**Parameters:**

*filename* [String](#) object that holds the name of the [Imageset](#) data file that is to be processed.

*resourceGroup* Resource group identifier to be passed to the resource manager. NB: This affects the imageset xml file only, the texture loaded may have its own group specified in the XML file.

**Returns:**

Nothing

**Exceptions:**

[FileIOException](#) thrown if something goes wrong while processing the file *filename*.

**6.130.2.27 void CEGUI::Imageset::setTexture (Texture \* *texture*)** [protected]

set the [Texture](#) object to be used by this [Imageset](#). Changing textures on an [Imageset](#) that is in use is not a good idea!

**Parameters:**

*texture* [Texture](#) object to be used by the [Imageset](#). The old texture is NOT disposed of, that is the clients responsibility.

**Returns:**

Nothing

**Exceptions:**

[NullObjectException](#) thrown if *texture* is NULL

**6.130.2.28 void CEGUI::Imageset::updateImageScalingFactors (void)** [protected]

Sets the scaling factor for all Images that are a part of this [Imageset](#).

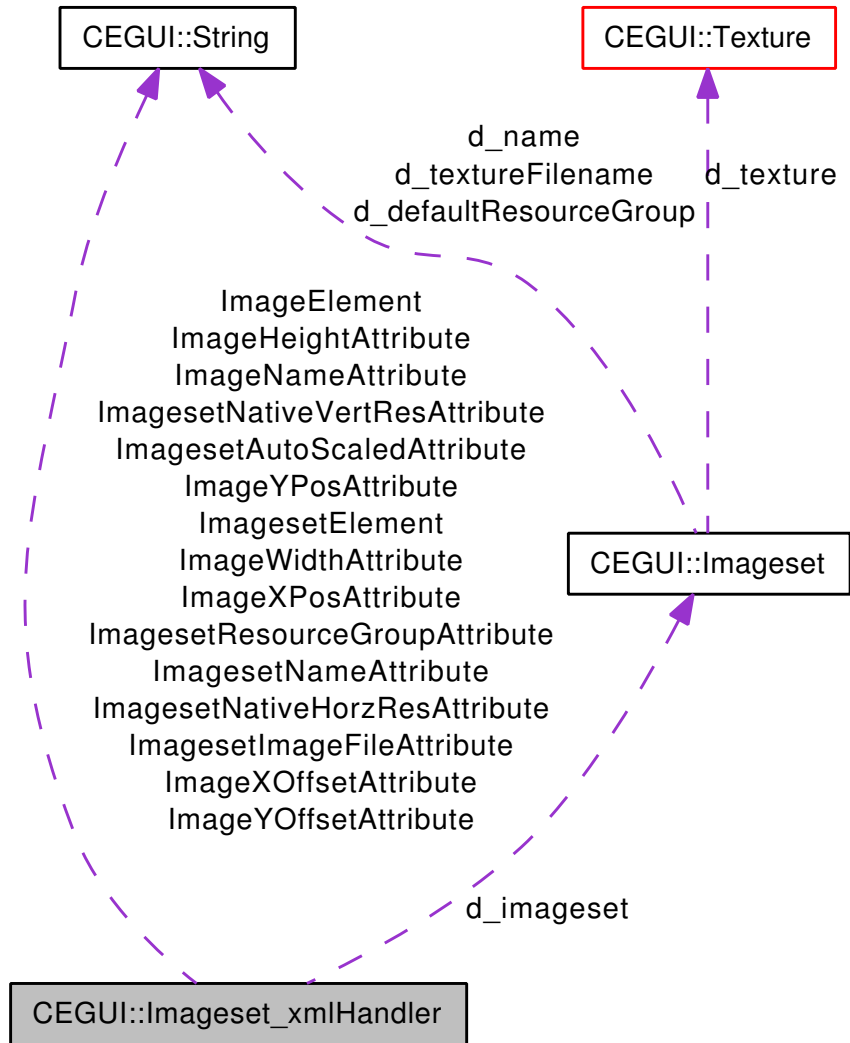
**Returns:**

Nothing.

## 6.131 CEGUI::Imageset\_xmlHandler Class Reference

Handler class used to parse the [Imageset](#) XML files using SAX2.

Collaboration diagram for CEGUI::Imageset\_xmlHandler:



### Public Member Functions

- [Imageset\\_xmlHandler](#) ([Imageset](#) \*imageset)  
*Constructor for Imageset::xmlHandler objects.*
- virtual [~Imageset\\_xmlHandler](#) (void)  
*Destructor for Imageset::xmlHandler objects.*
- virtual void [elementStart](#) (const [String](#) &element, const [XMLAttributes](#) &attributes)  
*document processing (only care about elements, schema validates format)*

- virtual void **elementEnd** (const [String](#) &element)
- [Imageset](#) \* **getImageset** (void) const

### 6.131.1 Detailed Description

Handler class used to parse the [Imageset](#) XML files using SAX2.

### 6.131.2 Constructor & Destructor Documentation

#### 6.131.2.1 CEGUI::Imageset\_xmlHandler::Imageset\_xmlHandler ([Imageset](#) \* *imageset*) [inline]

Constructor for Imageset::xmlHandler objects.

#### Parameters:

*imageset* Pointer to the [Imageset](#) object creating this xmlHandler object

## 6.132 CEGUI::ImagesetManager Class Reference

Class providing a shared library of [Imageset](#) objects to the system.

### Public Types

- typedef [ConstBaseIterator](#)< [ImagesetRegistry](#) > **ImagesetIterator**

### Public Member Functions

- [ImagesetManager](#) (void)  
*Constructor for [ImagesetManager](#) objects.*
- [~ImagesetManager](#) (void)  
*Destructor for [ImagesetManager](#) objects.*
- [Imageset](#) \* [createImageset](#) (const [String](#) &name, [Texture](#) \*texture)  
*Create a [Imageset](#) object with the given name and [Texture](#).*
- [Imageset](#) \* [createImageset](#) (const [String](#) &filename, const [String](#) &resourceGroup="")  
*Create an [Imageset](#) object from the specified file.*
- [Imageset](#) \* [createImagesetFromImageFile](#) (const [String](#) &name, const [String](#) &filename, const [String](#) &resourceGroup="")  
*Create an [Imageset](#) object from the specified image file. The [Imageset](#) will initially have a single image defined named "full\_image" which is an image that represents the entire area of the loaded image.*
- void [destroyImageset](#) (const [String](#) &name)  
*Destroys the [Imageset](#) with the specified name.*
- void [destroyImageset](#) ([Imageset](#) \*imageset)  
*Destroys the given [Imageset](#) object.*
- void [destroyAllImagesets](#) (void)  
*Destroys all [Imageset](#) objects registered in the system.*
- [Imageset](#) \* [getImageset](#) (const [String](#) &name) const  
*Returns a pointer to the [Imageset](#) object with the specified name.*
- bool [isImagesetPresent](#) (const [String](#) &name) const  
*Check for the existence of a named [Imageset](#).*
- void [notifyScreenResolution](#) (const [Size](#) &size)  
*Notify the [ImagesetManager](#) of the current (usually new) display resolution.*
- void [writeImagesetToStream](#) (const [String](#) &imageset, [OutStream](#) &out\_stream) const  
*Writes a full XML imageset for the specified [Imageset](#) to the given [OutStream](#).*
- [ImagesetIterator](#) [getIterator](#) (void) const  
*Return a [ImagesetManager::ImagesetIterator](#) object to iterate over the available [Imageset](#) objects.*

### 6.132.1 Detailed Description

Class providing a shared library of [Imageset](#) objects to the system.

The [ImagesetManager](#) is used to create, access, and destroy [Imageset](#) objects. The idea is that the [ImagesetManager](#) will function as a central repository for imagery used within the GUI system, and that such imagery can be accessed, via a unique name, by any interested party within the system.

### 6.132.2 Member Function Documentation

#### 6.132.2.1 [Imageset](#) \* CEGUI::ImagesetManager::createImageset (const String & *name*, Texture \* *texture*)

Create a [Imageset](#) object with the given name and [Texture](#).

The created [Imageset](#) will be of limited use, and will require one or more images to be defined for the set.

##### Parameters:

*name* [String](#) object containing the unique name for the [Imageset](#) being created.

*texture* [Texture](#) object to be associated with the [Imageset](#)

##### Returns:

Pointer to the newly created [Imageset](#) object

##### Exceptions:

[AlreadyExistsException](#) Thrown if an [Imageset](#) named *name* is already present in the system.

#### 6.132.2.2 [Imageset](#) \* CEGUI::ImagesetManager::createImageset (const String & *filename*, const String & *resourceGroup* = "")

Create an [Imageset](#) object from the specified file.

##### Parameters:

*filename* [String](#) object holding the name of the [Imageset](#) definition file which should be used to create the [Imageset](#)

*resourceGroup* Resource group identifier to be passed to the resource manager. NB: This affects the imageset xml file only, the texture loaded may have its own group specified in the XML file.

##### Returns:

Pointer to the newly created [Imageset](#) object

##### Exceptions:

[AlreadyExistsException](#) Thrown if an [Imageset](#) named *name* is already present in the system.

[FileIOException](#) Thrown if something goes wrong while processing the file *filename*.

**6.132.2.3 Imageset \* CEGUI::ImagesetManager::createImagesetFromImageFile (const String & name, const String & filename, const String & resourceGroup = "")**

Create an [Imageset](#) object from the specified image file. The [Imageset](#) will initially have a single image defined named "full\_image" which is an image that represents the entire area of the loaded image.

**Parameters:**

*name* [String](#) object containing the unique name for the [Imageset](#) being created.

*filename* [String](#) object holding the name of the image file to be loaded.

*resourceGroup* Resource group identifier to be passed to the resource manager when loading the image file.

**Returns:**

Pointer to the newly created [Imageset](#) object

**Exceptions:**

[AlreadyExistsException](#) Thrown if an [Imageset](#) named *name* is already present in the system.

[FileNotFoundException](#) Thrown if something goes wrong while reading the image file *filename*.

**6.132.2.4 void CEGUI::ImagesetManager::destroyImageset (const String & name)**

Destroys the [Imageset](#) with the specified name.

**Parameters:**

*name* [String](#) object containing the name of the [Imageset](#) to be destroyed. If no such [Imageset](#) exists, nothing happens.

**Returns:**

Nothing.

**6.132.2.5 void CEGUI::ImagesetManager::destroyImageset (Imageset \* imageset)**

Destroys the given [Imageset](#) object.

**Parameters:**

*imageset* Pointer to the [Imageset](#) to be destroyed. If no such [Imageset](#) exists, nothing happens.

**Returns:**

Nothing.

**6.132.2.6 void CEGUI::ImagesetManager::destroyAllImagesets (void)**

Destroys all [Imageset](#) objects registered in the system.

**Returns:**

Nothing



**6.132.2.7 Imageset \* CEGUI::ImagesetManager::getImageset (const String & name) const**

Returns a pointer to the [Imageset](#) object with the specified name.

**Parameters:**

*name* [String](#) object containing the name of the [Imageset](#) to return a pointer to

**Returns:**

Pointer to the requested [Imageset](#) object

**Exceptions:**

[UnknownObjectException](#) Thrown if no [Imageset](#) named *name* is present in within the system

**6.132.2.8 bool CEGUI::ImagesetManager::isImagesetPresent (const String & name) const**  
[inline]

Check for the existence of a named [Imageset](#).

**Parameters:**

*name* [String](#) object containing the name of the [Imageset](#) to look for

**Returns:**

true if an [Imageset](#) named *name* is presently loaded in the system, else false.

**6.132.2.9 void CEGUI::ImagesetManager::notifyScreenResolution (const Size & size)**

Notify the [ImagesetManager](#) of the current (usually new) display resolution.

**Parameters:**

*size* [Size](#) object describing the display resolution

**Returns:**

Nothing

**6.132.2.10 void CEGUI::ImagesetManager::writeImagesetToStream (const String & imageset, OutStream & out\_stream) const**

Writes a full XML imageset for the specified [Imageset](#) to the given OutStream.

**Parameters:**

*imageset* [String](#) holding the name of the [Imageset](#) to be written to the stream.

*out\_stream* OutStream (std::ostream based) object where data is to be sent.

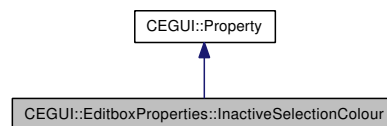
**Returns:**

Nothing.

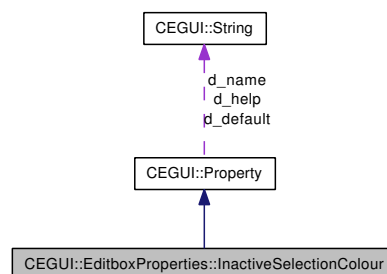
## 6.133 CEGUI::EditboxProperties::InactiveSelectionColour Class Reference

[Property](#) to access the [colour](#) used for rendering the selection highlight when the edit box is inactive.

Inheritance diagram for CEGUI::EditboxProperties::InactiveSelectionColour:



Collaboration diagram for CEGUI::EditboxProperties::InactiveSelectionColour:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.133.1 Detailed Description

[Property](#) to access the [colour](#) used for rendering the selection highlight when the edit box is inactive.

##### Usage:

- Name: [InactiveSelectionColour](#)
- Format: "aarrggb".

##### Where:

- aarrggb is the ARGB [colour](#) value to be used.

## 6.133.2 Member Function Documentation

### 6.133.2.1 String CEGUI::EditboxProperties::InactiveSelectionColour::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.133.2.2 void CEGUI::EditboxProperties::InactiveSelectionColour::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

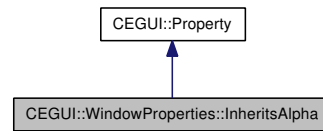
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

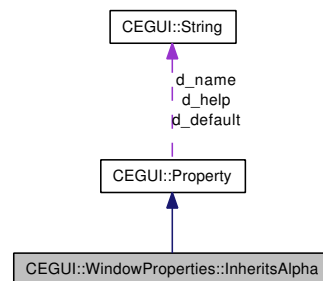
## 6.134 CEGUI::WindowProperties::InheritsAlpha Class Reference

[Property](#) to access window "Inherits Alpha" setting.

Inheritance diagram for CEGUI::WindowProperties::InheritsAlpha:



Collaboration diagram for CEGUI::WindowProperties::InheritsAlpha:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.134.1 Detailed Description

[Property](#) to access window "Inherits Alpha" setting.

This property offers access to the inherits alpha setting for the window.

**Usage:**

- Name: [InheritsAlpha](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the [Window](#) inherits alpha blend values from it's ancestors.
- "False" to indicate the [Window](#) does not inherit alpha blend values from it's ancestors.

## 6.134.2 Member Function Documentation

### 6.134.2.1 String CEGUI::WindowProperties::InheritsAlpha::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.134.2.2 void CEGUI::WindowProperties::InheritsAlpha::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

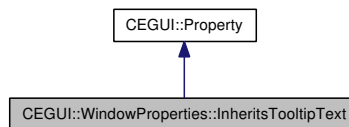
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

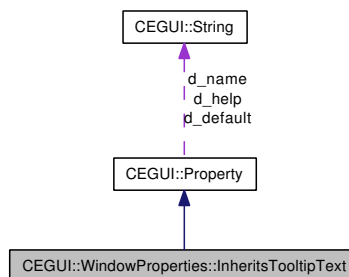
## 6.135 CEGUI::WindowProperties::InheritsTooltipText Class Reference

[Property](#) to access whether the window inherits its tooltip text from its parent when it has no tooltip text of its own.

Inheritance diagram for CEGUI::WindowProperties::InheritsTooltipText:



Collaboration diagram for CEGUI::WindowProperties::InheritsTooltipText:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.135.1 Detailed Description

[Property](#) to access whether the window inherits its tooltip text from its parent when it has no tooltip text of its own.

##### Usage:

- Name: [InheritsTooltipText](#)
- Format: "[text]".

##### Where [Text] is:

- "True" to indicate the [Window](#) inherits its tooltip text from its parent.
- "False" to indicate the [Window](#) does not inherit its tooltip text.

## 6.135.2 Member Function Documentation

### 6.135.2.1 String CEGUI::WindowProperties::InheritsTooltipText::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.135.2.2 void CEGUI::WindowProperties::InheritsTooltipText::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

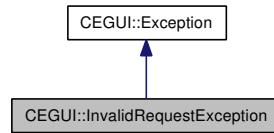
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

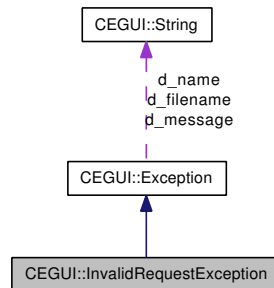
## 6.136 CEGUI::InvalidRequestException Class Reference

[Exception](#) class used when some impossible request was made for the current system state.

Inheritance diagram for CEGUI::InvalidRequestException:



Collaboration diagram for CEGUI::InvalidRequestException:



### Public Member Functions

- [InvalidRequestException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)

*Constructor that is responsible for logging the invalid request exception by calling the base class.*

#### 6.136.1 Detailed Description

[Exception](#) class used when some impossible request was made for the current system state.

#### 6.136.2 Constructor & Destructor Documentation

##### 6.136.2.1 CEGUI::InvalidRequestException::InvalidRequestException (const String & message, const String & file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the invalid request exception by calling the base class.

##### Parameters:

- message* [String](#) object describing the reason for the invalid request exception being thrown.
- filename* [String](#) object containing the name of the file where the invalid request exception occurred.
- line* Integer representing the line number where the invalid request exception occurred.



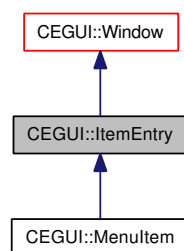
**Remarks:**

The invalid request exception name is automatically passed to the base class as "CEGUI::InvalidRequestException".

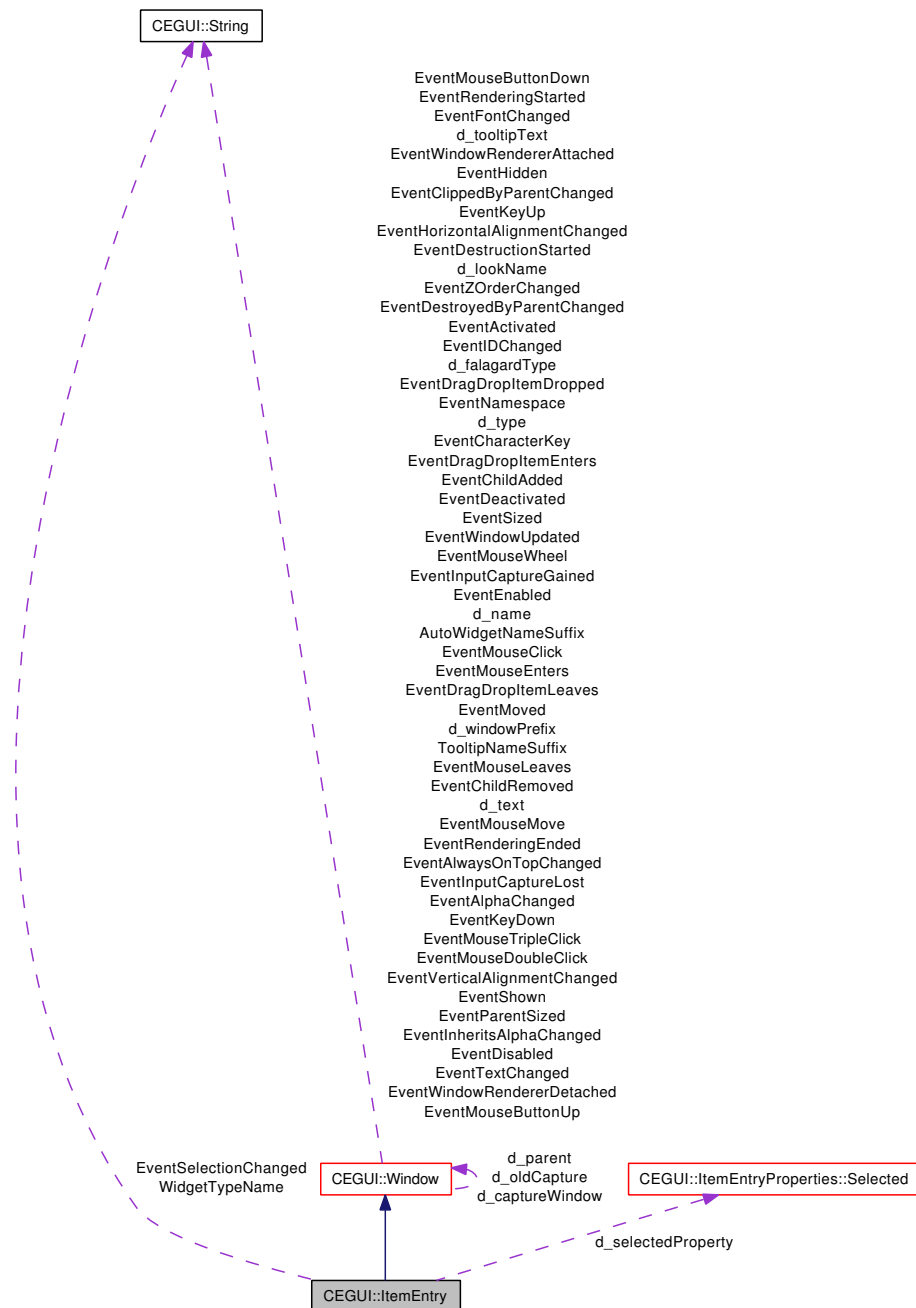
## 6.137 CEGUI::ItemEntry Class Reference

Base class for item type widgets.

Inheritance diagram for CEGUI::ItemEntry:



Collaboration diagram for CEGUI::ItemEntry:



## Public Member Functions

- `Size getItemPixelSize (void) const`

*Return the "optimal" size for the item.*

- `ItemListBase * getOwnerList (void) const`

*Returns a pointer to the owner [ItemListBase](#). 0 if there is none.*

- bool `isSelected` (void) const  
*Returns whether this item is selected or not.*
- bool `isSelectable` (void) const  
*Returns whether this item is selectable or not.*
- void `setSelected` (bool setting)  
*Sets the selection state of this item (on/off). If this item is not selectable this function does nothing.*
- void `select` (void)  
*Selects the item.*
- void `deselect` (void)  
*Deselects the item.*
- void `setSelected_impl` (bool state, bool notify)  
*Set the selection state for this ListItem. Internal version. Should NOT be used by client code.*
- void `setSelectable` (bool setting)  
*Sets whether this item will be selectable.*
- `ItemEntry` (const `String` &type, const `String` &name)  
*Constructor for `ItemEntry` objects.*
- virtual `~ItemEntry` (void)  
*Destructor for `ItemEntry` objects.*

## Static Public Attributes

- static const `String` `WidgetTypeName`  
*Window factory name.*
- static const `String` `EventSelectionChanged`  
*Event fired when selection state changes.*

## Protected Member Functions

- virtual bool `testClassName_impl` (const `String` &class\_name) const  
*Return the "optimal" size for the item.*
- virtual bool `validateWindowRenderer` (const `String` &name) const  
*Function used in checking if a `WindowRenderer` is valid for this window.*
- virtual void `onSelectionChanged` (`WindowEventArgs` &e)  
*Handles selection state changes.*

- virtual void [onMouseClicked](#) ([MouseEventArgs](#) &e)

*Handler called when a mouse button has been clicked (that is depressed and then released, within a specified time) within this window's area.*

## Protected Attributes

- [ItemListBase](#) \* [d\\_ownerList](#)  
< pointer to the owner [ItemListBase](#). 0 if there is none.
- bool [d\\_selected](#)  
'true' when the item is selectable.
- bool [d\\_selectable](#)

## Friends

- class [ItemListBase](#)

### 6.137.1 Detailed Description

Base class for item type widgets.

#### Todo

Fire events on selection / deselection. (Maybe selectable mode changed as well?)

### 6.137.2 Member Function Documentation

#### 6.137.2.1 Size CEGUI::ItemEntry::getItemPixelSize (void) const

Return the "optimal" size for the item.

#### Returns:

[Size](#) describing the size in pixel that this ItemEntry's content requires for non-clipped rendering

#### 6.137.2.2 void CEGUI::ItemEntry::setSelected (bool *setting*) [inline]

Sets the selection state of this item (on/off). If this item is not selectable this function does nothing.

#### Parameters:

*setting* 'true' to select the item. 'false' to deselect the item.

### 6.137.2.3 void CEGUI::ItemEntry::setSelectable (bool *setting*)

Sets whether this item will be selectable.

#### Parameters:

*setting* 'true' to allow this item to be selected. 'false' to disallow this item from ever being selected.

#### Note:

If the item is currently selectable and selected, calling this function with *setting* as 'false' will first deselect the item and then disable selectability.

### 6.137.2.4 virtual bool CEGUI::ItemEntry::testClassName\_impl (const String & *class\_name*) const [inline, protected, virtual]

Return the "optimal" size for the item.

#### Returns:

[Size](#) describing the size in pixel that this ItemEntry's content requires for non-clipped rendering

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

#### Parameters:

*class\_name* The class name that is to be checked.

#### Returns:

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::MenuItem](#).

### 6.137.2.5 virtual bool CEGUI::ItemEntry::validateWindowRenderer (const String & *name*) const [inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

#### Returns:

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

### 6.137.2.6 void CEGUI::ItemEntry::onMouseClicked (MouseEventArgs & *e*) [protected, virtual]

Handler called when a mouse button has been clicked (that is depressed and then released, within a specified time) within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.137.3 Member Data Documentation****6.137.3.1 ItemListBase\* CEGUI::ItemEntry::d\_ownerList** [protected]

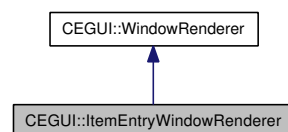
< pointer to the owner [ItemListBase](#). 0 if there is none.

'true' when the item is in the selected state, 'false' if not.

## 6.138 CEGUI::ItemEntryWindowRenderer Class Reference

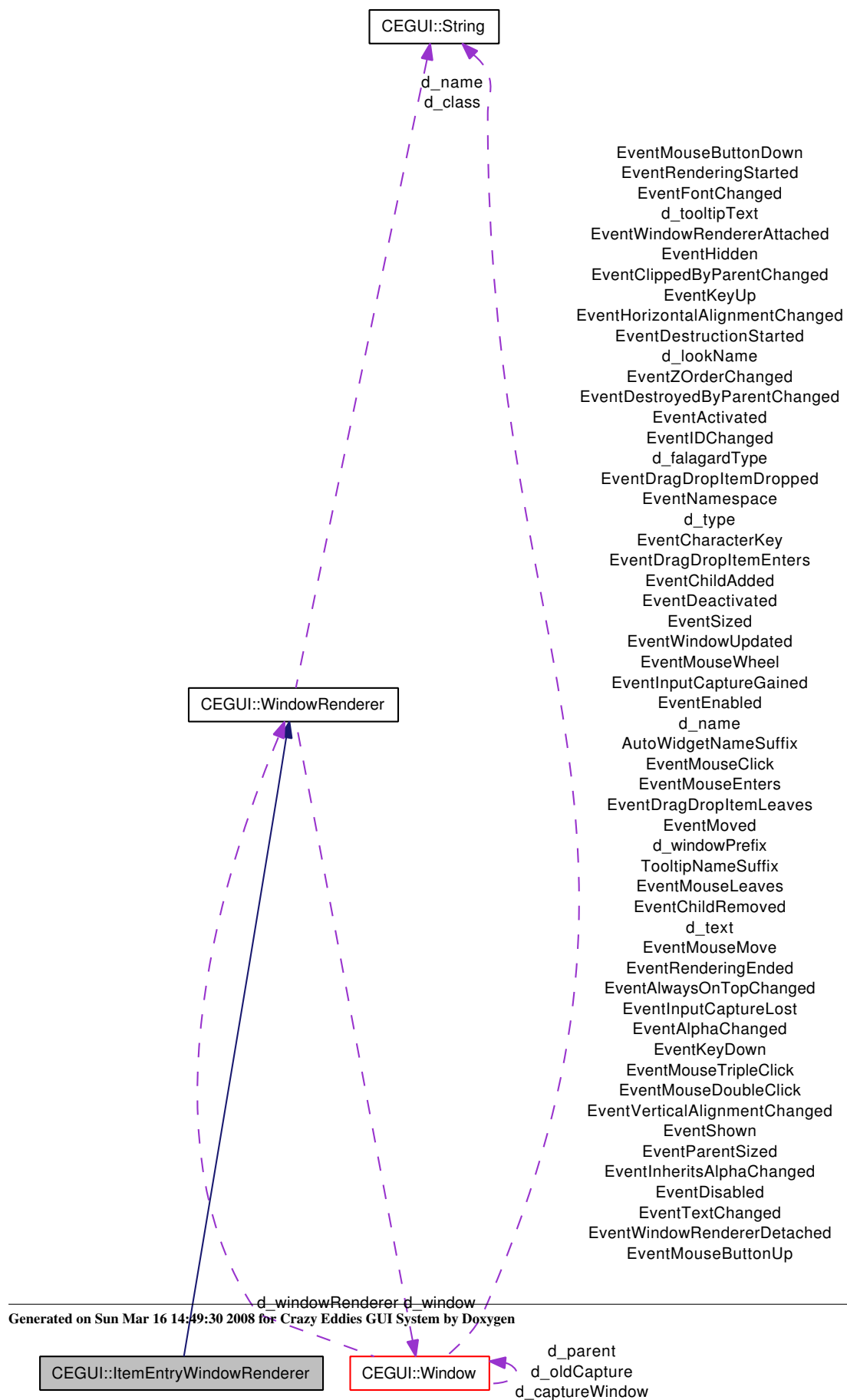
Base class for [ItemEntry](#) window renderer objects.

Inheritance diagram for CEGUI::ItemEntryWindowRenderer:





Collaboration diagram for CEGUI::ItemEntryWindowRenderer:



## Public Member Functions

- [ItemEntryWindowRenderer](#) (const [String](#) &name)

*Constructor.*

- virtual [Size](#) [getItemPixelSize](#) (void) const =0

*Return the "optimal" size for the item.*

### 6.138.1 Detailed Description

Base class for [ItemEntry](#) window renderer objects.

### 6.138.2 Member Function Documentation

#### 6.138.2.1 virtual [Size](#) CEGUI::ItemEntryWindowRenderer::getItemPixelSize (void) const [pure virtual]

Return the "optimal" size for the item.

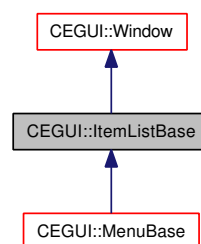
#### Returns:

[Size](#) describing the size in pixel that this ItemEntry's content requires for non-clipped rendering

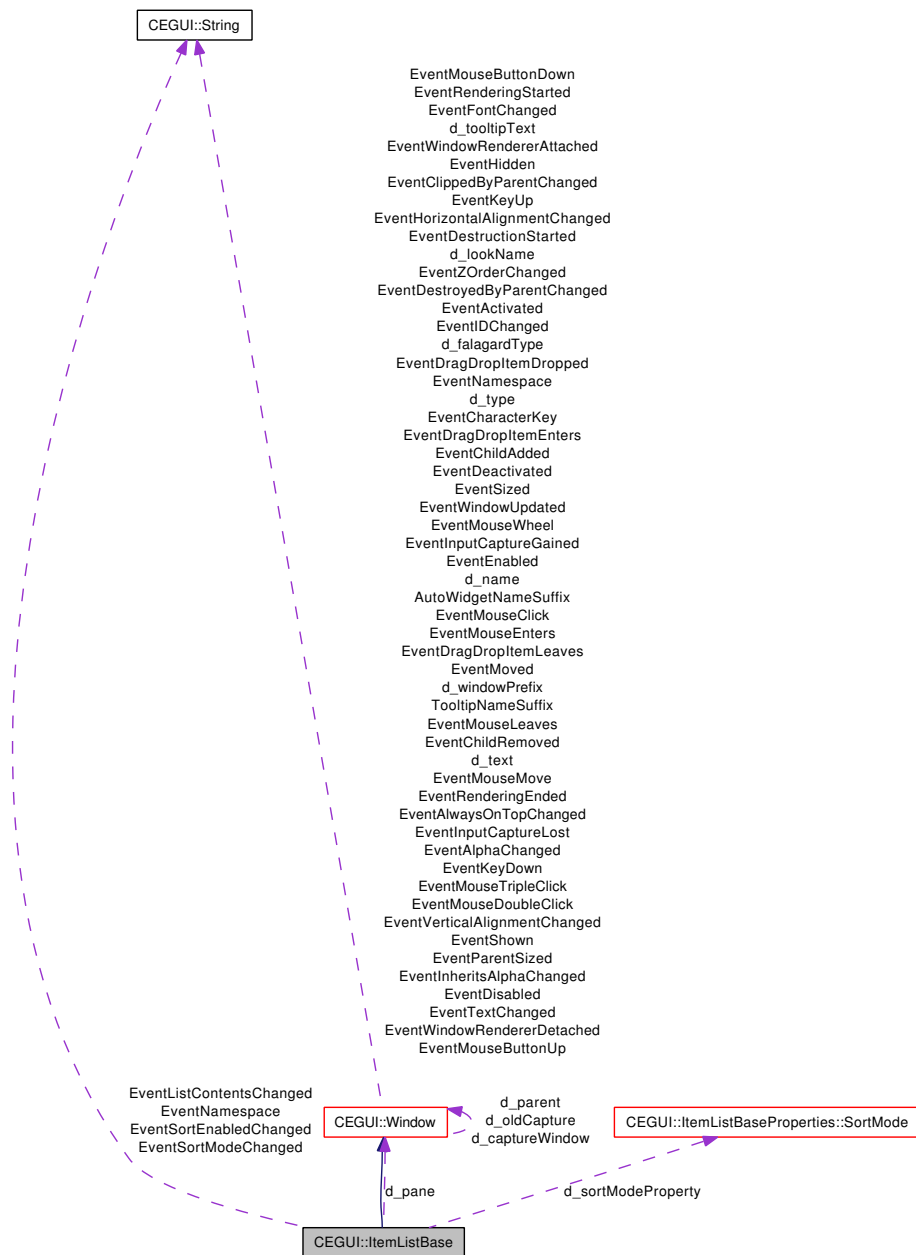
## 6.139 CEGUI::ItemListBase Class Reference

Base class for item list widgets.

Inheritance diagram for CEGUI::ItemListBase:



Collaboration diagram for CEGUI::ItemListBase:



## Public Types

- enum [SortMode](#) { **Ascending**, **Descending**, **UserSort** }

*Sort modes for [ItemListBase](#).*

- typedef bool(\* **SortCallback** )(const [ItemEntry](#) \*a, const [ItemEntry](#) \*b)

## Public Member Functions

- `size_t getItemCount` (void) const  
*Return number of items attached to the list.*
- `ItemEntry * getItemFromIndex` (size\_t index) const  
*Return the item at index position index.*
- `size_t getItemIndex` (const `ItemEntry` \*item) const  
*Return the index of `ItemEntry` item.*
- `ItemEntry * findItemWithText` (const `String` &text, const `ItemEntry` \*start\_item)  
*Search the list for an item with the specified text.*
- `bool isItemInList` (const `ItemEntry` \*item) const  
*Return whether the specified `ItemEntry` is in the List.*
- `bool isAutoResizeEnabled` () const  
*Return whether this window is automatically resized to fit its content.*
- `bool isSortEnabled` (void) const  
*Returns 'true' if the list is sorted.*
- `SortMode getSortMode` (void) const  
*Get sort mode.*
- `SortCallback getSortCallback` (void) const  
*Get user sorting callback.*
- `virtual void initialiseComponents` (void)  
*Initialise the `Window` based object ready for use.*
- `void resetList` (void)  
*Remove all items from the list.*
- `void addItem` (`ItemEntry` \*item)  
*Add the given `ItemEntry` to the list.*
- `void insertItem` (`ItemEntry` \*item, const `ItemEntry` \*position)  
*Insert an item into the list after a specified item already in the list.*
- `void removeItem` (`ItemEntry` \*item)  
*Removes the given item from the list. If the item is has the 'DestroyedByParent' property set to 'true', the item will be deleted.*
- `void handleUpdatedItemData` (bool resort=false)  
*Causes the list to update it's internal state after changes have been made to one or more attached `ItemEntry` objects.*
- `void setAutoResizeEnabled` (bool setting)

Set whether or not this *ItemListBase* widget should automatically resize to fit its content.

- virtual void *sizeToContent* (void)  
*Resize the ItemListBase to exactly fit the content that is attached to it. Return a Rect object describing, in un-clipped pixels, the window relative area that is to be used for rendering items.*
- virtual void *endInitialisation* (void)  
*Triggers a ListContentsChanged event. These are not fired during initialisation for optimization purposes.*
- virtual void *performChildWindowLayout* (void)  
*method called to perform extended laying out of attached child windows.*
- *Rect* *getItemRenderArea* (void) const  
*Return a Rect object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- *Window* \* *getContentPane* (void) const  
*Returns a pointer to the window that all items are directed too.*
- virtual void *notifyItemClicked* (*ItemEntry* \*li)  
*Notify this ItemListBase that the given item was just clicked. Internal function - NOT to be used from client code.*
- virtual void *notifyItemSelectState* (*ItemEntry* \*li, bool state)  
*Notify this ItemListBase that the given item just changed selection state. Internal function - NOT to be used from client code.*
- void *setSortEnabled* (bool setting)  
*Set whether the list should be sorted (by text).*
- void *setSortMode* (*SortMode* mode)  
*Set mode to be used when sorting the list.*
- void *setSortCallback* (*SortCallback* cb)  
*Set a user callback as sorting function.*
- void *sortList* (bool relayout=true)  
*Sort the list.*
- *ItemListBase* (const *String* &type, const *String* &name)  
*Constructor for ItemListBase base class.*
- virtual *~ItemListBase* (void)  
*Destructor for ItemListBase base class.*

## Static Public Attributes

- static const *String* *EventNamespace*  
*Namespace for global events.*

- static const [String](#) [EventListContentsChanged](#)  
*Event triggered when the contents of the list is changed.*
- static const [String](#) [EventSortEnabledChanged](#)  
*Event fired when the sort enabled state changes.*
- static const [String](#) [EventSortModeChanged](#)  
*Event fired when the sort mode changes.*

## Protected Types

- typedef std::vector< [ItemEntry](#) \* > [ItemEntryList](#)

## Protected Member Functions

- virtual void [sizeToContent\\_impl](#) (void)  
*Resize the [ItemListBase](#) to exactly fit the content that is attached to it. Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering items.*
- virtual [Size](#) [getContentSize](#) () const =0  
*Returns the [Size](#) in unclipped pixels of the content attached to this [ItemListBase](#) that is attached to it.*
- virtual void [layoutItemWidgets](#) ()=0  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- bool [resetList\\_impl](#) (void)  
*Remove all items from the list.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- SortCallback [getRealSortCallback](#) (void) const  
*Returns the SortCallback that's really going to be used for the sorting operation.*
- virtual void [onListContentsChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the list contents are changed.*
- virtual void [onSortEnabledChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when sorting gets enabled.*
- virtual void [onSortModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called internally when the sorting mode is changed.*

## Protected Attributes

- ItemEntryList [d\\_listItems](#)  
*list of items in the list.*
- bool [d\\_autoResize](#)  
*Pointer to the content pane (for items), 0 if we're not using one.*
- Window \* [d\\_pane](#)  
*True if this [ItemListBase](#) is sorted. False if not.*
- bool [d\\_sortEnabled](#)  
*The current sorting mode applied if sorting is enabled.*
- SortMode [d\\_sortMode](#)  
*The user sort callback or 0 if none.*
- SortCallback [d\\_sortCallback](#)  
*True if the list needs to be resorted.*
- bool [d\\_resort](#)

### 6.139.1 Detailed Description

Base class for item list widgets.

### 6.139.2 Member Function Documentation

#### 6.139.2.1 `size_t CEGUI::ItemListBase::getItemCount (void) const` [inline]

Return number of items attached to the list.

##### Returns:

the number of items currently attached to this list.

#### 6.139.2.2 `ItemEntry * CEGUI::ItemListBase::getItemFromIndex (size_t index) const`

Return the item at index position *index*.

##### Parameters:

*index* Zero based index of the item to be returned.

##### Returns:

Pointer to the [ItemEntry](#) at index position *index* in the list.

##### Exceptions:

[InvalidRequestException](#) thrown if *index* is out of range.



### 6.139.2.3 `size_t CEGUI::ItemListBase::getItemIndex (const ItemEntry * item) const`

Return the index of [ItemEntry](#) *item*.

#### Parameters:

*item* Pointer to a [ItemEntry](#) whos zero based index is to be returned.

#### Returns:

Zero based index indicating the position of [ItemEntry](#) *item* in the list.

#### Exceptions:

[InvalidRequestException](#) thrown if *item* is not attached to this list.

### 6.139.2.4 `ItemEntry * CEGUI::ItemListBase::findItemWithText (const String & text, const ItemEntry * start_item)`

Search the list for an item with the specified text.

#### Parameters:

*text* [String](#) object containing the text to be searched for.

*start\_item* [ItemEntry](#) where the search is to begin, the search will not include *item*. If *item* is NULL, the search will begin from the first item in the list.

#### Returns:

Pointer to the first [ItemEntry](#) in the list after *item* that has text matching *text*. If no item matches the criteria NULL is returned.

#### Exceptions:

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

### 6.139.2.5 `bool CEGUI::ItemListBase::isItemInList (const ItemEntry * item) const`

Return whether the specified [ItemEntry](#) is in the List.

#### Returns:

true if [ItemEntry](#) *item* is in the list, false if [ItemEntry](#) *item* is not in the list.

### 6.139.2.6 `bool CEGUI::ItemListBase::isAutoResizeEnabled () const` `[inline]`

Return whether this window is automatically resized to fit its content.

#### Returns:

true if automatic resizing is enabled, false if it is disabled.

**6.139.2.7 void CEGUI::ItemListBase::initialiseComponents (void) [virtual]**

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::ScrolledItemListBase](#).

**6.139.2.8 void CEGUI::ItemListBase::resetList (void)**

Remove all items from the list.

Note that this will cause items, which does not have the 'DestroyedByParent' property set to 'false', to be deleted.

**6.139.2.9 void CEGUI::ItemListBase::addItem (ItemEntry \* *item*)**

Add the given [ItemEntry](#) to the list.

**Parameters:**

*item* Pointer to the [ItemEntry](#) to be added to the list. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

**Returns:**

Nothing.

**6.139.2.10 void CEGUI::ItemListBase::insertItem (ItemEntry \* *item*, const ItemEntry \* *position*)**

Insert an item into the list after a specified item already in the list.

Note that if the list is sorted, the item may not end up in the requested position.

**Parameters:**

*item* Pointer to the [ItemEntry](#) to be inserted. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

*position* Pointer to a [ItemEntry](#) that *item* is to be inserted after. If this parameter is NULL, the item is inserted at the start of the list.

**Returns:**

Nothing.

**6.139.2.11 void CEGUI::ItemListBase::removeItem (ItemEntry \* *item*)**

Removes the given item from the list. If the item is has the 'DestroyedByParent' property set to 'true', the item will be deleted.

**Parameters:**

*item* Pointer to the [ItemEntry](#) that is to be removed. If *item* is not attached to this list then nothing will happen.

**Returns:**

Nothing.

**6.139.2.12 void CEGUI::ItemListBase::handleUpdatedItemData (bool *resort* = false)**

Causes the list to update it's internal state after changes have been made to one or more attached [ItemEntry](#) objects.

It should not be necessary to call this from client code, as the ItemEntries themselves call it if their parent is an [ItemListBase](#).

**Parameters:**

*resort* 'true' to redo the list sorting as well. 'false' to only do layout and perhaps auto resize. (defaults to 'false')

**Returns:**

Nothing.

**6.139.2.13 void CEGUI::ItemListBase::setAutoResizeEnabled (bool *setting*)**

Set whether or not this [ItemListBase](#) widget should automatically resize to fit its content.

**Parameters:**

*setting* Boolean value that if true enables automatic resizing, if false disables automatic resizing.

**Returns:**

Nothing.

**6.139.2.14 virtual void CEGUI::ItemListBase::sizeToContent (void) [inline, virtual]**

Resize the [ItemListBase](#) to exactly fit the content that is attached to it. Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering items.

**Returns:**

Nothing

**6.139.2.15 void CEGUI::ItemListBase::performChildWindowLayout (void) [virtual]**

method called to perform extended laying out of attached child windows.

The system may call this at various times (like when it is resized for example), and it may be invoked directly where required.

**Returns:**

Nothing.

Reimplemented from [CEGUI::Window](#).

**6.139.2.16 Rect CEGUI::ItemListBase::getItemRenderArea (void) const**

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the window relative area of the that is to be used for rendering the items.

**6.139.2.17 Window\* CEGUI::ItemListBase::getContentPane (void) const [inline]**

Returns a pointer to the window that all items are directed too.

**Returns:**

A pointer to the content pane window, or 'this' if children are added directly to this window.

**6.139.2.18 void CEGUI::ItemListBase::setSortMode (SortMode *mode*)**

Set mode to be used when sorting the list.

**Parameters:**

*mode* SortMode enum.

**6.139.2.19 void CEGUI::ItemListBase::setSortCallback (SortCallback *cb*)**

Set a user callback as sorting function.

**Parameters:**

*mode* SortCallback

**6.139.2.20 void CEGUI::ItemListBase::sortList (bool *relayout* = true)**

Sort the list.

**Parameters:**

*relayout* True if the item layout should be redone after the sorting. False to only sort the internal list.  
Nothing more.

This parameter defaults to true and should generally not be used in client code.

**6.139.2.21 void CEGUI::ItemListBase::sizeToContent\_impl (void) [protected, virtual]**

Resize the [ItemListBase](#) to exactly fit the content that is attached to it. Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering items.

**Returns:**

Nothing

**6.139.2.22 virtual Size CEGUI::ItemListBase::getContentSize () const [protected, pure virtual]**

Returns the [Size](#) in unclipped pixels of the content attached to this [ItemListBase](#) that is attached to it.

**Returns:**

[Size](#) object describing in unclipped pixels the size of the content ItemEntries attached to this menu.

Implemented in [CEGUI::ItemListbox](#), [CEGUI::Menubar](#), and [CEGUI::PopupMenu](#).

**6.139.2.23 virtual void CEGUI::ItemListBase::layoutItemWidgets () [protected, pure virtual]**

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the window relative area of the that is to be used for rendering the items.

Setup size and position for the item widgets attached to this [ItemListBase](#)

**Returns:**

Nothing.

Implemented in [CEGUI::ItemListbox](#), [CEGUI::Menubar](#), and [CEGUI::PopupMenu](#).

**6.139.2.24** `bool CEGUI::ItemListBase::resetList_impl (void)` [protected]

Remove all items from the list.

**Note:**

Note that this will cause items with the 'DestroyedByParent' property set to 'true', to be deleted.

**Returns:**

- true if the list contents were changed.
- false if the list contents were not changed (list already empty).

**6.139.2.25** `virtual bool CEGUI::ItemListBase::testClassName_impl (const String & class_name)`  
`const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::ItemListbox](#), [CEGUI::Menubar](#), [CEGUI::MenuBase](#), [CEGUI::PopupMenu](#), and [CEGUI::ScrolledItemListBase](#).

**6.139.2.26** `virtual bool CEGUI::ItemListBase::validateWindowRenderer (const String & name)`  
`const` [inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

## 6.139.3 Member Data Documentation

**6.139.3.1** `ItemEntryList CEGUI::ItemListBase::d_listItems` [protected]

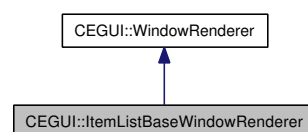
list of items in the list.

True if this [ItemListBase](#) widget should automatically resize to fit its content. False if not.

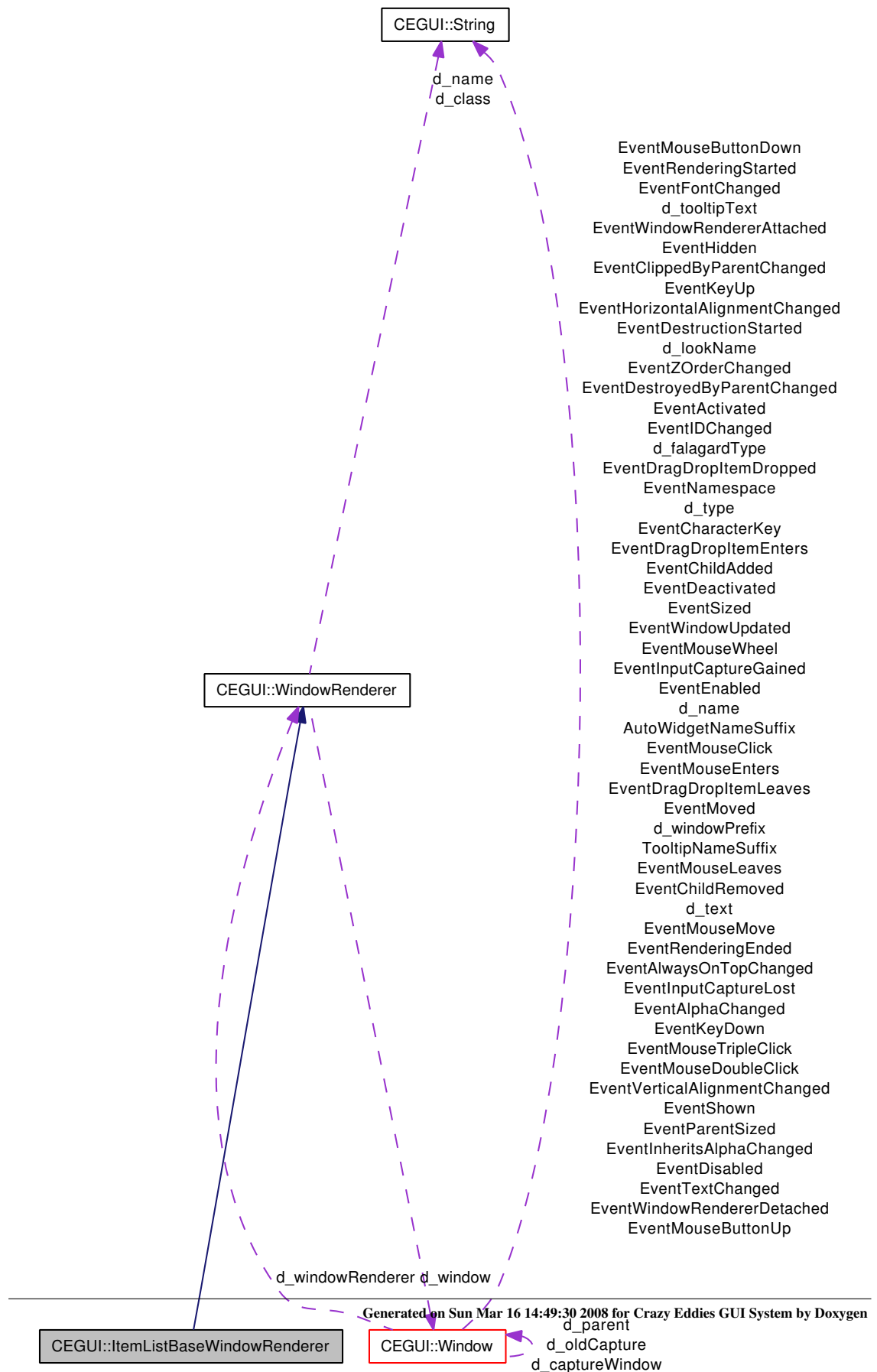
## 6.140 CEGUI::ItemListBaseWindowRenderer Class Reference

Base class for [ItemListBase](#) window renderer.

Inheritance diagram for CEGUI::ItemListBaseWindowRenderer:



Collaboration diagram for CEGUI::ItemListBaseWindowRenderer:





## Public Member Functions

- [ItemListBaseWindowRenderer](#) (const [String](#) &name)

*Constructor.*

- virtual [Rect](#) [getItemRenderArea](#) (void) const =0

*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*

### 6.140.1 Detailed Description

Base class for [ItemListBase](#) window renderer.

### 6.140.2 Member Function Documentation

#### 6.140.2.1 virtual [Rect](#) CEGUI::ItemListBaseWindowRenderer::getItemRenderArea (void) const [pure virtual]

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

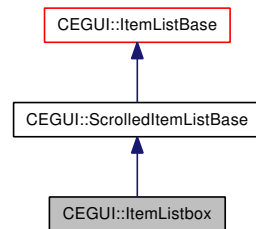
#### Returns:

[Rect](#) object describing the window relative area of the that is to be used for rendering the items.

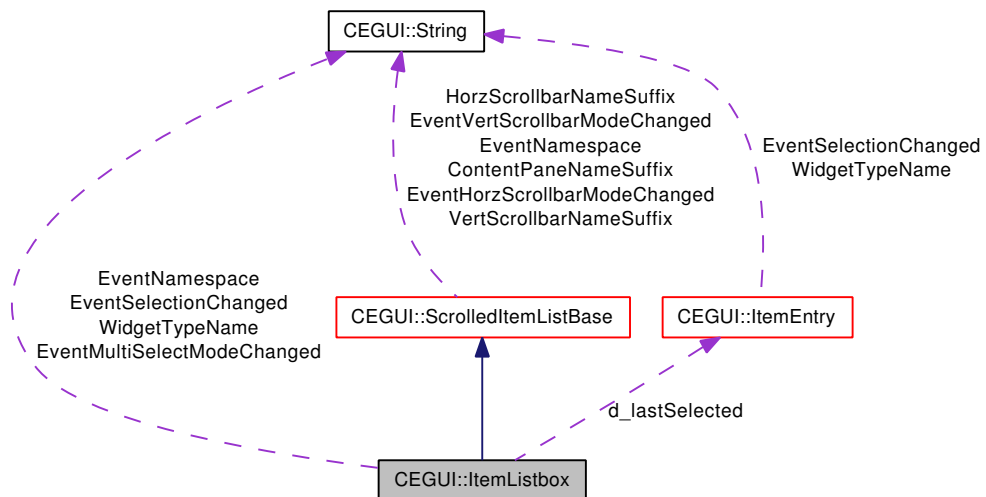
## 6.141 CEGUI::ItemListbox Class Reference

[ItemListbox](#) window class.

Inheritance diagram for CEGUI::ItemListbox:



Collaboration diagram for CEGUI::ItemListbox:



### Public Member Functions

- `size_t` [getSelectedCount](#) (void) const  
*Returns the number of selected items in this [ItemListbox](#).*
- `ItemEntry *` [getLastSelectedItem](#) (void) const  
*Returns a pointer to the last selected item.*
- `ItemEntry *` [getFirstSelectedItem](#) (size\_t start\_index=0) const  
*Returns a pointer to the first selected item.*
- `ItemEntry *` [getNextSelectedItem](#) (void) const  
*Returns a pointer to the next selected item relative to a previous call to [getFirstSelectedItem](#) or [getNextSelectedItem](#).*
- `ItemEntry *` [getNextSelectedItemAfter](#) (const [ItemEntry](#) \*start\_item) const

*Returns a pointer to the next selected item after the item 'start\_item' given.*

- bool [isMultiSelectEnabled](#) (void) const  
*Returns 'true' if multiple selections are allowed. 'false' if not.*
- bool [isItemSelected](#) (size\_t index) const  
*Returns 'true' if the item at the given index is selectable and currently selected.*
- void [setMultiSelectEnabled](#) (bool state)  
*Set whether or not multiple selections should be allowed.*
- void [clearAllSelections](#) (void)  
*Clears all selections.*
- void [selectRange](#) (size\_t a, size\_t z)  
*Select a range of items.*
- void [selectAllItems](#) (void)  
*Select all items. Does nothing if multiselect is disabled.*
- [ItemListbox](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for the [ItemListbox](#) base class constructor.*
- virtual [~ItemListbox](#) (void)  
*Destructor for the [ItemListbox](#) base class.*
- virtual void [layoutItemWidgets](#) ()  
*Setup size and position for the item widgets attached to this [ItemListbox](#).*
- virtual [Size](#) [getContentSize](#) () const  
*Returns the [Size](#) in unclipped pixels of the content attached to this [ItemListbox](#).*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [notifyItemClicked](#) ([ItemEntry](#) \*li)  
*Notify this [ItemListbox](#) that the given ListItem was just clicked. Internal function - not to be used from client code.*
- virtual void [notifyItemSelectState](#) ([ItemEntry](#) \*li, bool state)  
*Notify this [ItemListbox](#) that the given ListItem just changed selection state. Internal function - not to be used from client code.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*

- static const [String WidgetTypeName](#)  
*Window factory name.*
- static const [String EventSelectionChanged](#)  
*Event fired when the selection changes.*
- static const [String EventMultiSelectModeChanged](#)  
*Event fired when the multiselect mode changes.*

## Protected Member Functions

- [ItemEntry](#) \* [findSelectedItem](#) (size\_t start\_index) const  
*Returns a pointer to the first selected item starting the search from start\_index.*
- virtual void [onSelectionChanged](#) ([WindowEventArgs](#) &e)
- virtual void [onMultiSelectModeChanged](#) ([WindowEventArgs](#) &e)
- virtual void [onKeyDown](#) ([KeyEventArgs](#) &e)  
*Handler called when a key as been depressed while this window has input focus.*

## Protected Attributes

- bool [d\\_multiSelect](#)
- [ItemEntry](#) \* [d\\_lastSelected](#)  
*Controls whether multiple items can be selected simultaneously.*
- size\_t [d\\_nextSelectionIndex](#)  
*The last item that was selected.*

## Static Protected Attributes

- static [ItemListboxProperties::MultiSelect](#) [d\\_multiSelectProperty](#)

### 6.141.1 Detailed Description

[ItemListbox](#) window class.

### 6.141.2 Member Function Documentation

#### 6.141.2.1 [ItemEntry](#)\* [CEGUI::ItemListbox::getLastSelectedItem](#) (void) const [inline]

Returns a pointer to the last selected item.

#### Returns:

A pointer to the last selected item, 0 is none.

**6.141.2.2 ItemEntry \* CEGUI::ItemListbox::getFirstSelectedItem (size\_t start\_index = 0) const**

Returns a pointer to the first selected item.

**Parameters:**

*start\_index* The index where the search should begin. If omitted the search will begin with the first item.

**Returns:**

A pointer to the first selected item in the listbox. If no item is selected the return value is 0. If *start\_index* is out of bounds the return value is 0.

**Note:**

If multiselect is disabled then this does the equivalent of calling getLastSelectedItem. If multiselect is enabled it will search the array starting at *start\_index*

**6.141.2.3 ItemEntry \* CEGUI::ItemListbox::getNextSelectedItem (void) const**

Returns a pointer to the next selected item relative to a previous call to getFirstSelectedItem or getNextSelectedItem.

**Returns:**

A pointer to the next selected item. If there are no further selected items the return value is 0. If multiselect is disabled the return value is 0.

**Note:**

This member function will take on from where the last call to getFirstSelectedItem or getNextSelectedItem returned. So be sure to start with a call to getFirstSelectedItem.

This member function should be preferred over getNextSelectedItemAfter as it will perform better, especially on large lists.

**6.141.2.4 ItemEntry \* CEGUI::ItemListbox::getNextSelectedItemAfter (const ItemEntry \* start\_item) const**

Returns a pointer to the next selected item after the item 'start\_item' given.

**Note:**

This member function will search the array from the beginning and will be slow for large lists, it will not advance the internal counter used by getFirstSelectedItem and getNextSelectedItem either.

**6.141.2.5 void CEGUI::ItemListbox::selectRange (size\_t a, size\_t z)**

Select a range of items.

**Parameters:**

*a* Start item. (inclusive)  
*z* End item. (inclusive)

**6.141.2.6** `virtual bool CEGUI::ItemListbox::testClassName_impl (const String & class_name) const` `[inline, virtual]`

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::ScrolledItemListBase](#).

**6.141.2.7** `ItemEntry * CEGUI::ItemListbox::findSelectedItem (size_t start_index) const` `[protected]`

Returns a pointer to the first selected item starting the search from *start\_index*.

**Parameters:**

*start\_index* The index where the search should begin (inclusive)

**Returns:**

A pointer to the first selected item in the listbox found If no item is selected the return value is 0 If *start\_index* is out of bounds the return value is 0

**Note:**

This function advances the internal counter and is made for `getFirstSelectedItem` and `getNextSelectedItem`

**6.141.2.8** `void CEGUI::ItemListbox::onKeyDown (KeyEventArgs & e)` `[protected, virtual]`

Handler called when a key as been depressed while this window has input focus.

**Parameters:**

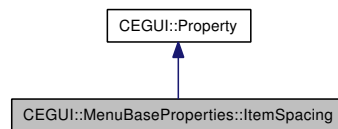
*e* [KeyEventArgs](#) object whose 'scancode' field is set to the `Key::Scan` value representing the key that was pressed, and whose 'sysKeys' field represents the combination of `SystemKey` that were active when the event was generated.

Reimplemented from [CEGUI::Window](#).

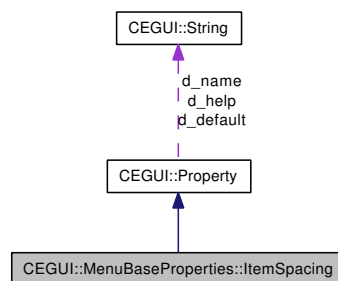
## 6.142 CEGUI::MenuBaseProperties::ItemSpacing Class Reference

[Property](#) to access the item spacing of the menu.

Inheritance diagram for CEGUI::MenuBaseProperties::ItemSpacing:



Collaboration diagram for CEGUI::MenuBaseProperties::ItemSpacing:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.142.1 Detailed Description

[Property](#) to access the item spacing of the menu.

##### Usage:

- Name: [ItemSpacing](#)
- Format: "[float]".

##### Where:

- [float] represents the item spacing of the menu.

## 6.142.2 Member Function Documentation

### 6.142.2.1 `String CEGUI::MenuBaseProperties::ItemSpacing::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.142.2.2 `void CEGUI::MenuBaseProperties::ItemSpacing::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

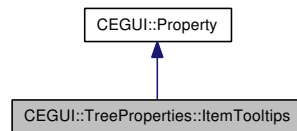
Implements [CEGUI::Property](#).



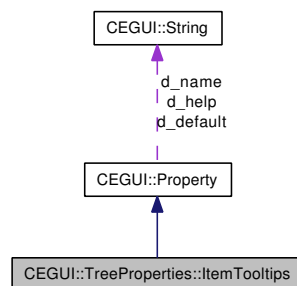
## 6.143 CEGUI::TreeProperties::ItemTooltips Class Reference

[Property](#) to access the show item tooltips setting of the list box.

Inheritance diagram for CEGUI::TreeProperties::ItemTooltips:



Collaboration diagram for CEGUI::TreeProperties::ItemTooltips:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.143.1 Detailed Description

[Property](#) to access the show item tooltips setting of the list box.

Usage:

- Name: [ItemTooltips](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate that the tooltip of the list box will be set by the item below the mouse pointer
- "False" to indicate that the list box has a static tooltip.

## 6.143.2 Member Function Documentation

### 6.143.2.1 `String CEGUI::TreeProperties::ItemTooltips::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.143.2.2 `void CEGUI::TreeProperties::ItemTooltips::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

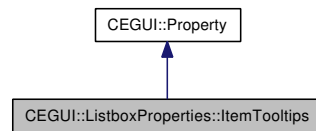
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

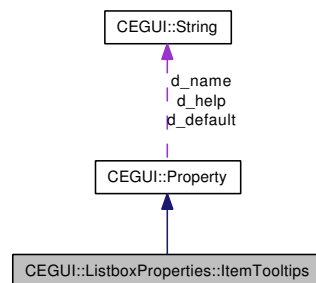
## 6.144 CEGUI::ListboxProperties::ItemTooltips Class Reference

[Property](#) to access the show item tooltips setting of the list box.

Inheritance diagram for CEGUI::ListboxProperties::ItemTooltips:



Collaboration diagram for CEGUI::ListboxProperties::ItemTooltips:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.144.1 Detailed Description

[Property](#) to access the show item tooltips setting of the list box.

**Usage:**

- Name: [ItemTooltips](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that the tooltip of the list box will be set by the item below the mouse pointer
- "False" to indicate that the list box has a static tooltip.

## 6.144.2 Member Function Documentation

### 6.144.2.1 `String CEGUI::ListboxProperties::ItemTooltips::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.144.2.2 `void CEGUI::ListboxProperties::ItemTooltips::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.145 CEGUI::Key Struct Reference

struct to give scope to scan code enumeration.

### Public Types

- enum **Scan** {
  - Escape** = 0x01, **One** = 0x02, **Two** = 0x03, **Three** = 0x04,
  - Four** = 0x05, **Five** = 0x06, **Six** = 0x07, **Seven** = 0x08,
  - Eight** = 0x09, **Nine** = 0x0A, **Zero** = 0x0B, **Minus** = 0x0C,
  - Equals** = 0x0D, **Backspace** = 0x0E, **Tab** = 0x0F, **Q** = 0x10,
  - W** = 0x11, **E** = 0x12, **R** = 0x13, **T** = 0x14,
  - Y** = 0x15, **U** = 0x16, **I** = 0x17, **O** = 0x18,
  - P** = 0x19, **LeftBracket** = 0x1A, **RightBracket** = 0x1B, **Return** = 0x1C,
  - LeftControl** = 0x1D, **A** = 0x1E, **S** = 0x1F, **D** = 0x20,
  - F** = 0x21, **G** = 0x22, **H** = 0x23, **J** = 0x24,
  - K** = 0x25, **L** = 0x26, **Semicolon** = 0x27, **Apostrophe** = 0x28,
  - Grave** = 0x29, **LeftShift** = 0x2A, **Backslash** = 0x2B, **Z** = 0x2C,
  - X** = 0x2D, **C** = 0x2E, **V** = 0x2F, **B** = 0x30,
  - N** = 0x31, **M** = 0x32, **Comma** = 0x33, **Period** = 0x34,
  - Slash** = 0x35, **RightShift** = 0x36, **Multiply** = 0x37, **LeftAlt** = 0x38,
  - Space** = 0x39, **Capital** = 0x3A, **F1** = 0x3B, **F2** = 0x3C,
  - F3** = 0x3D, **F4** = 0x3E, **F5** = 0x3F, **F6** = 0x40,
  - F7** = 0x41, **F8** = 0x42, **F9** = 0x43, **F10** = 0x44,
  - NumLock** = 0x45, **ScrollLock** = 0x46, **Numpad7** = 0x47, **Numpad8** = 0x48,
  - Numpad9** = 0x49, **Subtract** = 0x4A, **Numpad4** = 0x4B, **Numpad5** = 0x4C,
  - Numpad6** = 0x4D, **Add** = 0x4E, **Numpad1** = 0x4F, **Numpad2** = 0x50,
  - Numpad3** = 0x51, **Numpad0** = 0x52, **Decimal** = 0x53, **OEM\_102** = 0x56,
  - F11** = 0x57, **F12** = 0x58, **F13** = 0x64, **F14** = 0x65,
  - F15** = 0x66, **Kana** = 0x70, **ABNT\_C1** = 0x73, **Convert** = 0x79,
  - NoConvert** = 0x7B, **Yen** = 0x7D, **ABNT\_C2** = 0x7E, **NumpadEquals** = 0x8D,
  - PrevTrack** = 0x90, **At** = 0x91, **Colon** = 0x92, **Underline** = 0x93,
  - Kanji** = 0x94, **Stop** = 0x95, **AX** = 0x96, **Unlabeled** = 0x97,
  - NextTrack** = 0x99, **NumpadEnter** = 0x9C, **RightControl** = 0x9D, **Mute** = 0xA0,
  - Calculator** = 0xA1, **PlayPause** = 0xA2, **MediaStop** = 0xA4, **VolumeDown** = 0xAE,
  - VolumeUp** = 0xB0, **WebHome** = 0xB2, **NumpadComma** = 0xB3, **Divide** = 0xB5,
  - SysRq** = 0xB7, **RightAlt** = 0xB8, **Pause** = 0xC5, **Home** = 0xC7,
  - ArrowUp** = 0xC8, **PageUp** = 0xC9, **ArrowLeft** = 0xCB, **ArrowRight** = 0xCD,
  - End** = 0xCF, **ArrowDown** = 0xD0, **PageDown** = 0xD1, **Insert** = 0xD2,
  - Delete** = 0xD3, **LeftWindows** = 0xDB, **RightWindow** = 0xDC, **RightWindows** = 0xDC,
  - AppMenu** = 0xDD, **Power** = 0xDE, **Sleep** = 0xDF, **Wake** = 0xE3,

**WebSearch** = 0xE5, **WebFavorites** = 0xE6, **WebRefresh** = 0xE7, **WebStop** = 0xE8,  
**WebForward** = 0xE9, **WebBack** = 0xEA, **MyComputer** = 0xEB, **Mail** = 0xEC,  
**MediaSelect** = 0xED }

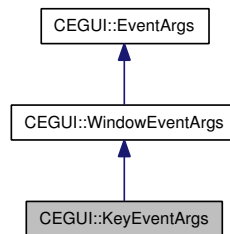
### 6.145.1 Detailed Description

struct to give scope to scan code enumeration.

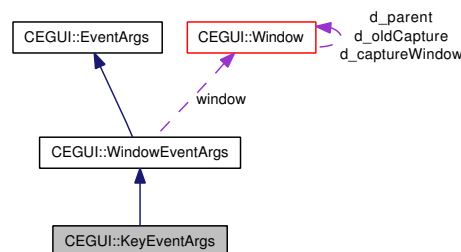
## 6.146 CEGUI::KeyEventArgs Class Reference

[EventArgs](#) based class that is used for objects passed to input event handlers concerning keyboard input.

Inheritance diagram for CEGUI::KeyEventArgs:



Collaboration diagram for CEGUI::KeyEventArgs:



### Public Member Functions

- **KeyEventArgs** ([Window](#) \*wnd)

### Public Attributes

- utf32 [codepoint](#)  
*utf32 codepoint for the key (only used for Character inputs).*
- Key::Scan [scancode](#)  
*Scan code of key that caused event (only used for key up & down inputs).*
- uint [sysKeys](#)  
*current state of the system keys and mouse buttons.*

#### 6.146.1 Detailed Description

[EventArgs](#) based class that is used for objects passed to input event handlers concerning keyboard input.

## 6.147 CEGUI::LayerSpecification Class Reference

Class that encapsulates a single layer of imagery.

### Public Member Functions

- [LayerSpecification](#) (uint priority)

*Constructor.*

- void [render](#) ([Window](#) &srcWindow, float base\_z, const [ColourRect](#) \*modcols=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const

*Render this layer.*

- void [render](#) ([Window](#) &srcWindow, const [Rect](#) &baseRect, float base\_z, const [ColourRect](#) \*modcols=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const

*Render this layer.*

- void [addSectionSpecification](#) (const [SectionSpecification](#) &section)

*Add a section specification to the layer.*

- void [clearSectionSpecifications](#) ()

*Clear all section specifications from this layer.*

- uint [getLayerPriority](#) () const

*Return the priority of this layer.*

- bool [operator<](#) (const [LayerSpecification](#) &other) const

- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const

*Writes an xml representation of this Layer to out\_stream.*

### 6.147.1 Detailed Description

Class that encapsulates a single layer of imagery.

### 6.147.2 Constructor & Destructor Documentation

#### 6.147.2.1 CEGUI::LayerSpecification::LayerSpecification (uint *priority*)

Constructor.

#### Parameters:

***priority*** Specifies the priority of the layer. Layers with higher priorities will be drawn on top of layers with lower priorities.



### 6.147.3 Member Function Documentation

**6.147.3.1** void CEGUI::LayerSpecification::render (Window & *srcWindow*, float *base\_z*, const ColourRect \* *modcols* = 0, const Rect \* *clipper* = 0, bool *clipToDisplay* = false) const

Render this layer.

**Parameters:**

*srcWindow* [Window](#) to use when calculating pixel values from [BaseDim](#) values.

*base\_z* base level z value to use for all imagery in the layer.

**Returns:**

Nothing.

**6.147.3.2** void CEGUI::LayerSpecification::render (Window & *srcWindow*, const Rect & *baseRect*, float *base\_z*, const ColourRect \* *modcols* = 0, const Rect \* *clipper* = 0, bool *clipToDisplay* = false) const

Render this layer.

**Parameters:**

*srcWindow* [Window](#) to use when calculating pixel values from [BaseDim](#) values.

*baseRect* [Rect](#) to use when calculating pixel values from [BaseDim](#) values.

*base\_z* base level z value to use for all imagery in the layer.

**Returns:**

Nothing.

**6.147.3.3** void CEGUI::LayerSpecification::addSectionSpecification (const SectionSpecification & *section*)

Add a section specification to the layer.

A section specification is a reference to a named [ImagerySection](#) within the WidgetLook.

**Parameters:**

*section* [SectionSpecification](#) object describing the section that should be added to this layer.

**Returns:**

Nothing.

**6.147.3.4** void CEGUI::LayerSpecification::clearSectionSpecifications ()

Clear all section specifications from this layer.

**Returns:**

Nothing.

**6.147.3.5    `uint CEGUI::LayerSpecification::getLayerPriority () const`**

Return the priority of this layer.

**Returns:**

uint value describing the priority of this [LayerSpecification](#).

**6.147.3.6    `void CEGUI::LayerSpecification::writeXMLToStream (XMLSerializer & xml_stream) const`**

Writes an xml representation of this Layer to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

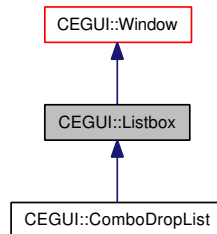
**Returns:**

Nothing.

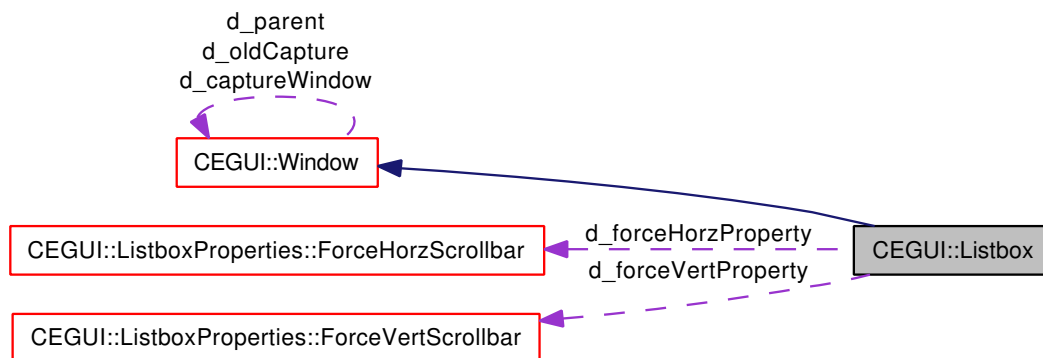
## 6.148 CEGUI::Listbox Class Reference

Base class for standard [Listbox](#) widget.

Inheritance diagram for CEGUI::Listbox:



Collaboration diagram for CEGUI::Listbox:



### Public Member Functions

- `size_t` [getItemCount](#) (void) const  
*Return number of items attached to the list box.*
- `size_t` [getSelectedCount](#) (void) const  
*Return the number of selected items in the list box.*
- `ListboxItem *` [getFirstSelectedItem](#) (void) const  
*Return a pointer to the first selected item.*
- `ListboxItem *` [getNextSelected](#) (const `ListboxItem *`start\_item) const  
*Return a pointer to the next selected item after item start\_item.*
- `ListboxItem *` [getListboxItemFromIndex](#) (size\_t index) const  
*Return the item at index position index.*
- `size_t` [getItemIndex](#) (const `ListboxItem *`item) const  
*Return the index of [ListboxItem](#) item.*

- bool **isSortEnabled** (void) const  
*return whether list sorting is enabled*
- bool **isMultiselectEnabled** (void) const  
*return whether multi-select is enabled*
- bool **isItemTooltipsEnabled** (void) const
- bool **isItemSelected** (size\_t index) const  
*return whether the string at index position index is selected*
- **ListboxItem** \* **findItemWithText** (const **String** &text, const **ListboxItem** \*start\_item)  
*Search the list for an item with the specified text.*
- bool **isListboxItemInList** (const **ListboxItem** \*item) const  
*Return whether the specified **ListboxItem** is in the List.*
- bool **isVertScrollbarAlwaysShown** (void) const  
*Return whether the vertical scroll bar is always shown.*
- bool **isHorzScrollbarAlwaysShown** (void) const  
*Return whether the horizontal scroll bar is always shown.*
- virtual void **initialiseComponents** (void)  
*Initialise the **Window** based object ready for use.*
- void **resetList** (void)  
*Remove all items from the list.*
- void **addItem** (**ListboxItem** \*item)  
*Add the given **ListboxItem** to the list.*
- void **insertItem** (**ListboxItem** \*item, const **ListboxItem** \*position)  
*Insert an item into the list box after a specified item already in the list.*
- void **removeItem** (const **ListboxItem** \*item)  
*Removes the given item from the list box. If the item is has the auto delete state set, the item will be deleted.*
- void **clearAllSelections** (void)  
*Clear the selected state for all items.*
- void **setSortingEnabled** (bool setting)  
*Set whether the list should be sorted.*
- void **setMultiselectEnabled** (bool setting)  
*Set whether the list should allow multiple selections or just a single selection.*
- void **setShowVertScrollbar** (bool setting)  
*Set whether the vertical scroll bar should always be shown.*
- void **setShowHorzScrollbar** (bool setting)

*Set whether the horizontal scroll bar should always be shown.*

- void **setItemTooltipsEnabled** (bool setting)
- void **setItemSelectState** ([ListboxItem](#) \*item, bool state)  
*Set the select state of an attached [ListboxItem](#).*
- void **setItemSelectState** (size\_t item\_index, bool state)  
*Set the select state of an attached [ListboxItem](#).*
- void **handleUpdatedItemData** (void)  
*Causes the list box to update it's internal state after changes have been made to one or more attached [ListboxItem](#) objects.*
- void **ensureItemIsVisible** (size\_t item\_index)  
*Ensure the item at the specified index is visible within the list box.*
- void **ensureItemIsVisible** (const [ListboxItem](#) \*item)  
*Ensure the item at the specified index is visible within the list box.*
- virtual [Rect](#) **getListRenderArea** (void) const  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- [Scrollbar](#) \* **getVertScrollbar** () const  
*Return a pointer to the vertical scrollbar component widget for this [Listbox](#).*
- [Scrollbar](#) \* **getHorzScrollbar** () const  
*Return a pointer to the horizontal scrollbar component widget for this [Listbox](#).*
- float **getTotalItemsHeight** (void) const  
*Return the sum of all item heights.*
- float **getWidestItemWidth** (void) const  
*Return the width of the widest item.*
- [Listbox](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Listbox](#) base class.*
- virtual **~Listbox** (void)  
*Destructor for [Listbox](#) base class.*

## Static Public Attributes

- static const [String](#) **EventNamespace**  
*Namespace for global events.*
- static const [String](#) **WidgetTypeName**  
*[Window](#) factory name.*

- static const [String EventListContentsChanged](#)  
*Event triggered when the contents of the list is changed.*
- static const [String EventSelectionChanged](#)  
*Event triggered when there is a change to the currently selected item(s).*
- static const [String EventSortModeChanged](#)  
*Event triggered when the sort mode setting changes.*
- static const [String EventMultiselectModeChanged](#)  
*Event triggered when the multi-select mode setting changes.*
- static const [String EventVertScrollbarModeChanged](#)  
*Event triggered when the vertical scroll bar 'force' setting changes.*
- static const [String EventHorzScrollbarModeChanged](#)  
*Event triggered when the horizontal scroll bar 'force' setting changes.*
- static const [String VertScrollbarNameSuffix](#)  
*Widget name suffix for the vertical scrollbar component.*
- static const [String HorzScrollbarNameSuffix](#)  
*Widget name suffix for the horizontal scrollbar component.*

## Protected Types

- typedef std::vector< [ListboxItem](#) \* > **LBItemList**

## Protected Member Functions

- void [configureScrollbars](#) (void)  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- void [selectRange](#) (size\_t start, size\_t end)  
*select all strings between positions start and end. (inclusive) including end.*
- bool [clearAllSelections\\_impl](#) (void)  
*Clear the selected state for all items (implementation).*
- [ListboxItem](#) \* [getItemAtPoint](#) (const [Point](#) &pt) const  
*Return the [ListboxItem](#) under the given window local pixel co-ordinate.*
- bool [resetList\\_impl](#) (void)  
*Remove all items from the list.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const

*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

- bool `handle_scrollChange` (const `EventArgs` &args)  
*Internal handler that is triggered when the user interacts with the scrollbars.*
- virtual bool `validateWindowRenderer` (const `String` &name) const  
*Function used in checking if a `WindowRenderer` is valid for this window.*
- void `resortList` ()  
*Causes the internal list to be (re)sorted.*
- virtual void `onListContentsChanged` (`WindowEventArgs` &e)  
*Handler called internally when the list contents are changed.*
- virtual void `onSelectionChanged` (`WindowEventArgs` &e)  
*Handler called internally when the currently selected item or items changes.*
- virtual void `onSortModeChanged` (`WindowEventArgs` &e)  
*Handler called internally when the sort mode setting changes.*
- virtual void `onMultiselectModeChanged` (`WindowEventArgs` &e)  
*Handler called internally when the multi-select mode setting changes.*
- virtual void `onVertScrollbarModeChanged` (`WindowEventArgs` &e)  
*Handler called internally when the forced display of the vertical scroll bar setting changes.*
- virtual void `onHorzScrollbarModeChanged` (`WindowEventArgs` &e)  
*Handler called internally when the forced display of the horizontal scroll bar setting changes.*
- virtual void `onSized` (`WindowEventArgs` &e)  
*Handler called when the window's size changes.*
- virtual void `onMouseButtonDown` (`MouseEventArgs` &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void `onMouseWheel` (`MouseEventArgs` &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*
- virtual void `onMouseMove` (`MouseEventArgs` &e)  
*Handler called when the mouse cursor has been moved within this window's area.*

## Protected Attributes

- bool `d_sorted`  
*true if list is sorted*
- bool `d_multiselect`  
*true if multi-select is enabled*

- bool [d\\_forceVertScroll](#)  
*true if vertical scrollbar should always be displayed*
- bool [d\\_forceHorzScroll](#)  
*true if horizontal scrollbar should always be displayed*
- bool [d\\_itemTooltips](#)  
*true if each item should have an individual tooltip*
- [LBItemList](#) [d\\_listItems](#)  
*list of items in the list box.*
- [ListBoxItem](#) \* [d\\_lastSelected](#)  
*holds pointer to the last selected item (used in range selections)*

## Friends

- class [ListBoxWindowRenderer](#)

### 6.148.1 Detailed Description

Base class for standard [ListBox](#) widget.

### 6.148.2 Member Function Documentation

#### 6.148.2.1 `size_t CEGUI::ListBox::getItemCount (void) const` [inline]

Return number of items attached to the list box.

##### Returns:

the number of items currently attached to this list box.

#### 6.148.2.2 `size_t CEGUI::ListBox::getSelectedCount (void) const`

Return the number of selected items in the list box.

##### Returns:

Total number of attached items that are in the selected state.

#### 6.148.2.3 `ListBoxItem * CEGUI::ListBox::getFirstSelectedItem (void) const`

Return a pointer to the first selected item.

##### Returns:

Pointer to a [ListBoxItem](#) based object that is the first selected item in the list. will return NULL if no item is selected.



**6.148.2.4 ListboxItem \* CEGUI::Listbox::getNextSelected (const ListboxItem \* start\_item) const**

Return a pointer to the next selected item after item *start\_item*.

**Parameters:**

*start\_item* Pointer to the [ListboxItem](#) where the search for the next selected item is to begin. If this parameter is NULL, the search will begin with the first item in the list box.

**Returns:**

Pointer to a [ListboxItem](#) based object that is the next selected item in the list after the item specified by *start\_item*. Will return NULL if no further items were selected.

**Exceptions:**

[InvalidRequestException](#) thrown if *start\_item* is not attached to this list box.

**6.148.2.5 ListboxItem \* CEGUI::Listbox::getListboxItemFromIndex (size\_t index) const**

Return the item at index position *index*.

**Parameters:**

*index* Zero based index of the item to be returned.

**Returns:**

Pointer to the [ListboxItem](#) at index position *index* in the list box.

**Exceptions:**

[InvalidRequestException](#) thrown if *index* is out of range.

**6.148.2.6 size\_t CEGUI::Listbox::getItemIndex (const ListboxItem \* item) const**

Return the index of [ListboxItem](#) *item*.

**Parameters:**

*item* Pointer to a [ListboxItem](#) whos zero based index is to be returned.

**Returns:**

Zero based index indicating the position of [ListboxItem](#) *item* in the list box.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.148.2.7 bool CEGUI::Listbox::isSortEnabled (void) const [inline]**

return whether list sorting is enabled

**Returns:**

true if the list is sorted, false if the list is not sorted

**6.148.2.8 bool CEGUI::Listbox::isMultiselectEnabled (void) const** `[inline]`

return whether multi-select is enabled

**Returns:**

true if multi-select is enabled, false if multi-select is not enabled.

**6.148.2.9 bool CEGUI::Listbox::isItemSelected (size\_t *index*) const**

return whether the string at index position *index* is selected

**Parameters:**

*index* Zero based index of the item to be examined.

**Returns:**

true if the item at *index* is selected, false if the item at *index* is not selected.

**Exceptions:**

*InvalidRequestException* thrown if *index* is out of range.

**6.148.2.10 ListboxItem \* CEGUI::Listbox::findItemWithText (const String & *text*, const ListboxItem \* *start\_item*)**

Search the list for an item with the specified text.

**Parameters:**

*text* *String* object containing the text to be searched for.

*start\_item* *ListboxItem* where the search is to begin, the search will not include *item*. If *item* is NULL, the search will begin from the first item in the list.

**Returns:**

Pointer to the first *ListboxItem* in the list after *item* that has text matching *text*. If no item matches the criteria NULL is returned.

**Exceptions:**

*InvalidRequestException* thrown if *item* is not attached to this list box.

**6.148.2.11 bool CEGUI::Listbox::isListboxItemInList (const ListboxItem \* *item*) const**

Return whether the specified *ListboxItem* is in the List.

**Returns:**

true if *ListboxItem* *item* is in the list, false if *ListboxItem* *item* is not in the list.

**6.148.2.12 bool CEGUI::Listbox::isVertScrollbarAlwaysShown (void) const**

Return whether the vertical scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.148.2.13 bool CEGUI::Listbox::isHorzScrollbarAlwaysShown (void) const**

Return whether the horizontal scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.148.2.14 void CEGUI::Listbox::initialiseComponents (void) [virtual]**

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::ComboDropList](#).

**6.148.2.15 void CEGUI::Listbox::resetList (void)**

Remove all items from the list.

Note that this will cause 'AutoDelete' items to be deleted.

**6.148.2.16 void CEGUI::Listbox::addItem (ListboxItem \* item)**

Add the given [ListboxItem](#) to the list.

**Parameters:**

*item* Pointer to the [ListboxItem](#) to be added to the list. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

**Returns:**

Nothing.

**6.148.2.17 void CEGUI::Listbox::insertItem (ListboxItem \* *item*, const ListboxItem \* *position*)**

Insert an item into the list box after a specified item already in the list.

Note that if the list is sorted, the item may not end up in the requested position.

**Parameters:**

*item* Pointer to the [ListboxItem](#) to be inserted. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

*position* Pointer to a [ListboxItem](#) that *item* is to be inserted after. If this parameter is NULL, the item is inserted at the start of the list.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if no [ListboxItem](#) *position* is attached to this list box.

**6.148.2.18 void CEGUI::Listbox::removeItem (const ListboxItem \* *item*)**

Removes the given item from the list box. If the item has the auto delete state set, the item will be deleted.

**Parameters:**

*item* Pointer to the [ListboxItem](#) that is to be removed. If *item* is not attached to this list box then nothing will happen.

**Returns:**

Nothing.

**6.148.2.19 void CEGUI::Listbox::clearAllSelections (void)**

Clear the selected state for all items.

**Returns:**

Nothing.

**6.148.2.20 void CEGUI::Listbox::setSortingEnabled (bool *setting*)**

Set whether the list should be sorted.

**Parameters:**

*setting* true if the list should be sorted, false if the list should not be sorted.

**Returns:**

Nothing.

**6.148.2.21 void CEGUI::Listbox::setMultiselectEnabled (bool *setting*)**

Set whether the list should allow multiple selections or just a single selection.

**Parameters:**

*setting* true if the widget should allow multiple items to be selected, false if the widget should only allow a single selection.

**Returns:**

Nothing.

**6.148.2.22 void CEGUI::Listbox::setShowVertScrollbar (bool *setting*)**

Set whether the vertical scroll bar should always be shown.

**Parameters:**

*setting* true if the vertical scroll bar should be shown even when it is not required. false if the vertical scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.148.2.23 void CEGUI::Listbox::setShowHorzScrollbar (bool *setting*)**

Set whether the horizontal scroll bar should always be shown.

**Parameters:**

*setting* true if the horizontal scroll bar should be shown even when it is not required. false if the horizontal scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.148.2.24 void CEGUI::Listbox::setItemSelectState (ListboxItem \* *item*, bool *state*)**

Set the select state of an attached [ListboxItem](#).

This is the recommended way of selecting and deselecting items attached to a list box as it respects the multi-select mode setting. It is possible to modify the setting on ListboxItems directly, but that approach does not respect the settings of the list box.

**Parameters:**

*item* The [ListboxItem](#) to be affected. This item must be attached to the list box.

*state* true to select the item, false to de-select the item.

**Returns:**

Nothing.

**Exceptions:**

*[InvalidRequestException](#)* thrown if *item* is not attached to this list box.

**6.148.2.25 void CEGUI::Listbox::setItemSelectState (size\_t item\_index, bool state)**

Set the select state of an attached [ListboxItem](#).

This is the recommended way of selecting and deselecting items attached to a list box as it respects the multi-select mode setting. It is possible to modify the setting on ListboxItems directly, but that approach does not respect the settings of the list box.

**Parameters:**

*item\_index* The zero based index of the [ListboxItem](#) to be affected. This must be a valid index (0 <= index < [getItemCount\(\)](#))

*state* true to select the item, false to de-select the item.

**Returns:**

Nothing.

**Exceptions:**

*[InvalidRequestException](#)* thrown if *item\_index* is out of range for the list box

**6.148.2.26 void CEGUI::Listbox::handleUpdatedItemData (void)**

Causes the list box to update it's internal state after changes have been made to one or more attached [ListboxItem](#) objects.

Client code must call this whenever it has made any changes to [ListboxItem](#) objects already attached to the list box. If you are just adding items, or removed items to update them prior to re-adding them, there is no need to call this method.

**Returns:**

Nothing.

**6.148.2.27 void CEGUI::Listbox::ensureItemIsVisible (size\_t item\_index)**

Ensure the item at the specified index is visible within the list box.

**Parameters:**

*item\_index* Zero based index of the item to be made visible in the list box. If this value is out of range, the list is always scrolled to the bottom.

**Returns:**

Nothing.

**6.148.2.28 void CEGUI::Listbox::ensureItemIsVisible (const ListboxItem \* *item*)**

Ensure the item at the specified index is visible within the list box.

**Parameters:**

*item* Pointer to the [ListboxItem](#) to be made visible in the list box.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.148.2.29 Rect CEGUI::Listbox::getListRenderArea (void) const [virtual]**

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

**6.148.2.30 Scrollbar \* CEGUI::Listbox::getVertScrollbar () const**

Return a pointer to the vertical scrollbar component widget for this [Listbox](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the vertical [Scrollbar](#) component does not exist.

**6.148.2.31 Scrollbar \* CEGUI::Listbox::getHorzScrollbar () const**

Return a pointer to the horizontal scrollbar component widget for this [Listbox](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the horizontal [Scrollbar](#) component does not exist.

**6.148.2.32 void CEGUI::Listbox::configureScrollbars (void) [protected]**

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

display required integrated scroll bars according to current state of the list box and update their values.

**6.148.2.33 bool CEGUI::Listbox::clearAllSelections\_impl (void) [protected]**

Clear the selected state for all items (implementation).

**Returns:**

true if some selections were cleared, false nothing was changed.

**6.148.2.34 ListboxItem \* CEGUI::Listbox::getItemAtPoint (const Point & *pt*) const [protected]**

Return the [ListboxItem](#) under the given window local pixel co-ordinate.

**Returns:**

[ListboxItem](#) that is under window pixel co-ordinate *pt*, or NULL if no item is under that position.

**6.148.2.35 bool CEGUI::Listbox::resetList\_impl (void) [protected]**

Remove all items from the list.

**Note:**

Note that this will cause 'AutoDelete' items to be deleted.

**Returns:**

- true if the list contents were changed.
- false if the list contents were not changed (list already empty).

**6.148.2.36 virtual bool CEGUI::Listbox::testClassName\_impl (const String & *class\_name*) const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.



**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::ComboDropList](#).

**6.148.2.37** `virtual bool CEGUI::Listbox::validateWindowRenderer (const String & name) const`  
[inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.148.2.38** `void CEGUI::Listbox::onSized (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's size changes.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.148.2.39** `void CEGUI::Listbox::onMouseDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::ComboDropList](#).

**6.148.2.40** `void CEGUI::Listbox::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.148.2.41** `void CEGUI::Listbox::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

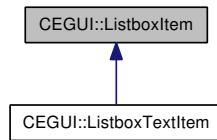
Reimplemented from [CEGUI::Window](#).

Reimplemented in [CEGUI::ComboDropList](#).

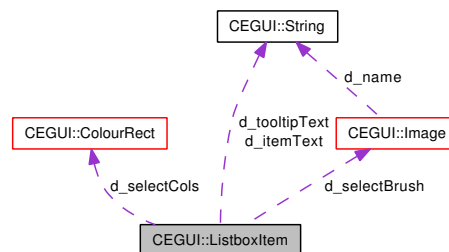
## 6.149 CEGUI::ListboxItem Class Reference

Base class for list box items.

Inheritance diagram for CEGUI::ListboxItem:



Collaboration diagram for CEGUI::ListboxItem:



### Public Member Functions

- **ListboxItem** (const **String** &text, uint item\_id=0, void \*item\_data=0, bool disabled=false, bool auto\_delete=true)  
*base class constructor*
- virtual **~ListboxItem** (void)  
*base class destructor*
- const **String** & **getText** (void) const  
*return the text string set for this list box item.*
- const **String** & **getTooltipText** (void) const
- uint **getID** (void) const  
*Return the current ID assigned to this list box item.*
- void \* **getUserData** (void) const  
*Return the pointer to any client assigned user data attached to this list box item.*
- bool **isSelected** (void) const  
*return whether this item is selected.*
- bool **isDisabled** (void) const  
*return whether this item is disabled.*
- bool **isAutoDeleted** (void) const

*return whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.*

- `const Window * getOwnerWindow () const`  
*Get the owner window for this [ListboxItem](#).*
- `ColourRect getSelectionColours (void) const`  
*Return the current colours used for selection highlighting.*
- `const Image * getSelectionBrushImage (void) const`  
*Return the current selection highlighting brush.*
- `void setText (const String &text)`  
*set the text string for this list box item.*
- `void setTooltipText (const String &text)`
- `void setID (uint item_id)`  
*Set the ID assigned to this list box item.*
- `void setUserData (void *item_data)`  
*Set the client assigned user data attached to this list box item.*
- `void setSelected (bool setting)`  
*set whether this item is selected.*
- `void setDisabled (bool setting)`  
*set whether this item is disabled.*
- `void setAutoDeleted (bool setting)`  
*Set whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.*
- `void setOwnerWindow (const Window *owner)`  
*Set the owner window for this [ListboxItem](#). This is called by all the list box widgets when an item is added or inserted.*
- `void setSelectionColours (const ColourRect &cols)`  
*Set the colours used for selection highlighting.*
- `void setSelectionColours (colour top_left_colour, colour top_right_colour, colour bottom_left_colour, colour bottom_right_colour)`  
*Set the colours used for selection highlighting.*
- `void setSelectionColours (colour col)`  
*Set the colours used for selection highlighting.*
- `void setSelectionBrushImage (const Image *image)`  
*Set the selection highlighting brush image.*
- `void setSelectionBrushImage (const String &imageset, const String &image)`

*Set the selection highlighting brush image.*

- virtual [Size](#) [getPixelSize](#) (void) const =0  
*Return the rendered pixel size of this list box item.*
- virtual void [draw](#) (const [Vector3](#) &position, float alpha, const [Rect](#) &clipper) const =0  
*Draw the list box item in its current state.*
- virtual void [draw](#) ([RenderCache](#) &cache, const [Rect](#) &targetRect, float zBase, float alpha, const [Rect](#) \*clipper) const =0
- virtual bool [operator<](#) (const [ListBoxItem](#) &rhs) const  
*Less-than operator, compares item texts.*
- virtual bool [operator>](#) (const [ListBoxItem](#) &rhs) const  
*Greater-than operator, compares item texts.*

## Static Public Attributes

- static const [colour](#) [DefaultSelectionColour](#) = 0xFF4444AA  
*Default selection brush [colour](#).*

## Protected Member Functions

- [ColourRect](#) [getModulateAlphaColourRect](#) (const [ColourRect](#) &cols, float alpha) const  
*Return a [ColourRect](#) object describing the colours in cols after having their alpha component modulated by the value alpha.*
- [colour](#) [calculateModulatedAlphaColour](#) ([colour](#) col, float alpha) const  
*Return a [colour](#) value describing the [colour](#) specified by col after having its alpha component modulated by the value alpha.*

## Protected Attributes

- [String](#) [d\\_itemText](#)  
*Text for this list box item. If not rendered, this is still used for list sorting.*
- [String](#) [d\\_tooltipText](#)  
*Text for the individual tooltip of this item.*
- uint [d\\_itemID](#)  
*ID code assigned by client code. This has no meaning within the GUI system.*
- void \* [d\\_itemData](#)  
*Pointer to some client code data. This has no meaning within the GUI system.*
- bool [d\\_selected](#)

*true if this item is selected. false if the item is not selected.*

- bool [d\\_disabled](#)

*true if this item is disabled. false if the item is not disabled.*

- bool [d\\_autoDelete](#)

*true if the system should destroy this item, false if client code will destroy the item.*

- const [Window](#) \* [d\\_owner](#)

*Pointer to the window that owns this item.*

- [ColourRect](#) [d\\_selectCols](#)

*Colours used for selection highlighting.*

- const [Image](#) \* [d\\_selectBrush](#)

*Image used for rendering selection.*

### 6.149.1 Detailed Description

Base class for list box items.

### 6.149.2 Member Function Documentation

#### 6.149.2.1 `const String& CEGUI::ListboxItem::getText (void) const` [inline]

return the text string set for this list box item.

Note that even if the item does not render text, the text string can still be useful, since it is used for sorting list box items.

##### Returns:

[String](#) object containing the current text for the list box item.

#### 6.149.2.2 `uint CEGUI::ListboxItem::getID (void) const` [inline]

Return the current ID assigned to this list box item.

Note that the system does not make use of this value, client code can assign any meaning it wishes to the ID.

##### Returns:

ID code currently assigned to this list box item

**6.149.2.3 void\* CEGUI::ListboxItem::getUserData (void) const** [inline]

Return the pointer to any client assigned user data attached to this list box item.

Note that the system does not make use of this data, client code can assign any meaning it wishes to the attached data.

**Returns:**

Pointer to the currently assigned user data.

**6.149.2.4 bool CEGUI::ListboxItem::isSelected (void) const** [inline]

return whether this item is selected.

**Returns:**

true if the item is selected, false if the item is not selected.

**6.149.2.5 bool CEGUI::ListboxItem::isDisabled (void) const** [inline]

return whether this item is disabled.

**Returns:**

true if the item is disabled, false if the item is enabled.

**6.149.2.6 bool CEGUI::ListboxItem::isAutoDeleted (void) const** [inline]

return whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.

**Returns:**

true if the item object will be deleted by the system when the list box it is attached to is destroyed, or when the item is removed from the list. false if client code must destroy the item after it is removed from the list.

**6.149.2.7 const Window\* CEGUI::ListboxItem::getOwnerWindow () const** [inline]

Get the owner window for this [ListboxItem](#).

The owner of a [ListboxItem](#) is typically set by the list box widgets when an item is added or inserted.

**Returns:**

Pointer to the window that is considered the owner of this [ListboxItem](#).

**6.149.2.8 ColourRect CEGUI::ListboxItem::getSelectionColours (void) const** [inline]

Return the current colours used for selection highlighting.

**Returns:**

[ColourRect](#) object describing the currently set colours

**6.149.2.9 const Image\* CEGUI::ListboxItem::getSelectionBrushImage (void) const** [inline]

Return the current selection highlighting brush.

**Returns:**

Pointer to the [Image](#) object currently used for selection highlighting.

**6.149.2.10 void CEGUI::ListboxItem::setText (const String & text)** [inline]

set the text string for this list box item.

Note that even if the item does not render text, the text string can still be useful, since it is used for sorting list box items.

**Parameters:**

*text* [String](#) object containing the text to set for the list box item.

**Returns:**

Nothing.

**6.149.2.11 void CEGUI::ListboxItem::setID (uint item\_id)** [inline]

Set the ID assigned to this list box item.

Note that the system does not make use of this value, client code can assign any meaning it wishes to the ID.

**Parameters:**

*item\_id* ID code to be assigned to this list box item

**Returns:**

Nothing.

**6.149.2.12 void CEGUI::ListboxItem::setUserData (void \* item\_data)** [inline]

Set the client assigned user data attached to this list box item.

Note that the system does not make use of this data, client code can assign any meaning it wishes to the attached data.



**Parameters:**

*item\_data* Pointer to the user data to attach to this list item.

**Returns:**

Nothing.

**6.149.2.13 void CEGUI::ListboxItem::setSelected (bool *setting*) [inline]**

set whether this item is selected.

**Parameters:**

*setting* true if the item is selected, false if the item is not selected.

**Returns:**

Nothing.

**6.149.2.14 void CEGUI::ListboxItem::setDisabled (bool *setting*) [inline]**

set whether this item is disabled.

**Parameters:**

*setting* true if the item is disabled, false if the item is enabled.

**Returns:**

Nothing.

**6.149.2.15 void CEGUI::ListboxItem::setAutoDeleted (bool *setting*) [inline]**

Set whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.

**Parameters:**

*setting* true if the item object should be deleted by the system when the list box it is attached to is destroyed, or when the item is removed from the list. false if client code will destroy the item after it is removed from the list.

**Returns:**

Nothing.

**6.149.2.16 void CEGUI::ListboxItem::setOwnerWindow (const Window \* *owner*) [inline]**

Set the owner window for this [ListboxItem](#). This is called by all the list box widgets when an item is added or inserted.

**Parameters:**

*owner* Ponter to the window that should be considered the owner of this [ListBoxItem](#).

**Returns:**

Nothing

**6.149.2.17 void CEGUI::ListBoxItem::setSelectionColours (const ColourRect & cols) [inline]**

Set the colours used for selection highlighting.

**Parameters:**

*cols* [ColourRect](#) object describing the colours to be used.

**Returns:**

Nothing.

**6.149.2.18 void CEGUI::ListBoxItem::setSelectionColours (colour *top\_left\_colour*, colour *top\_right\_colour*, colour *bottom\_left\_colour*, colour *bottom\_right\_colour*)**

Set the colours used for selection highlighting.

**Parameters:**

*top\_left\_colour* Colour (as ARGB value) to be applied to the top-left corner of the selection area.

*top\_right\_colour* Colour (as ARGB value) to be applied to the top-right corner of the selection area.

*bottom\_left\_colour* Colour (as ARGB value) to be applied to the bottom-left corner of the selection area.

*bottom\_right\_colour* Colour (as ARGB value) to be applied to the bottom-right corner of the selection area.

**Returns:**

Nothing.

**6.149.2.19 void CEGUI::ListBoxItem::setSelectionColours (colour *col*) [inline]**

Set the colours used for selection highlighting.

**Parameters:**

*col* [colour](#) value to be used when rendering.

**Returns:**

Nothing.

**6.149.2.20 void CEGUI::ListboxItem::setSelectionBrushImage (const Image \* *image*)**  
[inline]

Set the selection highlighting brush image.

**Parameters:**

*image* Pointer to the [Image](#) object to be used for selection highlighting.

**Returns:**

Nothing.

**6.149.2.21 void CEGUI::ListboxItem::setSelectionBrushImage (const String & *imageset*, const String & *image*)**

Set the selection highlighting brush image.

**Parameters:**

*imageset* Name of the imageset containing the image to be used.

*image* Name of the image to be used

**Returns:**

Nothing.

**6.149.2.22 virtual Size CEGUI::ListboxItem::getPixelSize (void) const** [pure virtual]

Return the rendered pixel size of this list box item.

**Returns:**

[Size](#) object describing the size of the list box item in pixels.

Implemented in [CEGUI::ListboxTextItem](#).

**6.149.2.23 virtual void CEGUI::ListboxItem::draw (const Vector3 & *position*, float *alpha*, const Rect & *clipper*) const** [pure virtual]

Draw the list box item in its current state.

**Parameters:**

*position* Vecor3 object describing the upper-left corner of area that should be rendered in to for the draw operation.

*alpha* Alpha value to be used when rendering the item (between 0.0f and 1.0f).

*clipper* [Rect](#) object describing the clipping rectangle for the draw operation.

**Returns:**

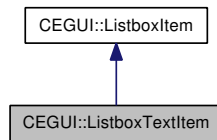
Nothing.

Implemented in [CEGUI::ListboxTextItem](#).

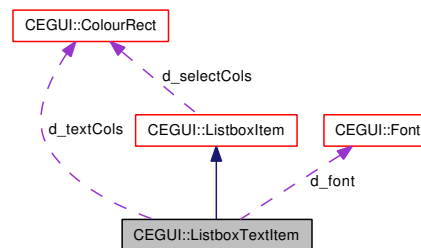
## 6.150 CEGUI::ListboxTextItem Class Reference

Class used for textual items in a list box.

Inheritance diagram for CEGUI::ListboxTextItem:



Collaboration diagram for CEGUI::ListboxTextItem:



### Public Member Functions

- [ListboxTextItem](#) (const [String](#) &text, uint item\_id=0, void \*item\_data=0, bool disabled=false, bool auto\_delete=true)  
*base class constructor*
- virtual [~ListboxTextItem](#) (void)  
*base class destructor*
- [Font](#) \* [getFont](#) (void) const  
*Return a pointer to the font being used by this [ListboxTextItem](#).*
- [ColourRect](#) [getTextColours](#) (void) const  
*Return the current colours used for text rendering.*
- void [setFont](#) ([Font](#) \*font)  
*Set the font to be used by this [ListboxTextItem](#).*
- void [setFont](#) (const [String](#) &font\_name)  
*Set the font to be used by this [ListboxTextItem](#).*
- void [setTextColours](#) (const [ColourRect](#) &cols)  
*Set the colours used for text rendering.*
- void [setTextColours](#) ([colour](#) top\_left\_colour, [colour](#) top\_right\_colour, [colour](#) bottom\_left\_colour, [colour](#) bottom\_right\_colour)

*Set the colours used for text rendering.*

- void [setTextColours](#) (colour col)

*Set the colours used for text rendering.*

- [Size](#) [getPixelSize](#) (void) const

*Return the rendered pixel size of this list box item.*

- void [draw](#) (const [Vector3](#) &position, float alpha, const [Rect](#) &clipper) const

*Draw the list box item in its current state.*

- void [draw](#) ([RenderCache](#) &cache, const [Rect](#) &targetRect, float zBase, float alpha, const [Rect](#) \*clipper) const

## Static Public Attributes

- static const colour [DefaultTextColour](#) = 0xFFFFFFFF

*Default text colour.*

## Protected Attributes

- [ColourRect](#) d\_textCols

*Colours used for rendering the text.*

- [Font](#) \* d\_font

*Font used for rendering text.*

### 6.150.1 Detailed Description

Class used for textual items in a list box.

### 6.150.2 Member Function Documentation

#### 6.150.2.1 [Font](#) \* CEGUI::ListboxTextItem::getFont (void) const

Return a pointer to the font being used by this [ListboxTextItem](#).

This method will try a number of places to find a font to be used. If no font can be found, NULL is returned.

#### Returns:

[Font](#) to be used for rendering this item

**6.150.2.2 ColourRect CEGUI::ListboxTextItem::getTextColours (void) const** [inline]

Return the current colours used for text rendering.

**Returns:**

[ColourRect](#) object describing the currently set colours

**6.150.2.3 void CEGUI::ListboxTextItem::setFont (Font \*font)** [inline]

Set the font to be used by this [ListboxTextItem](#).

**Parameters:**

*font* [Font](#) to be used for rendering this item

**Returns:**

Nothing

**6.150.2.4 void CEGUI::ListboxTextItem::setFont (const String &font\_name)**

Set the font to be used by this [ListboxTextItem](#).

**Parameters:**

*font\_name* [String](#) object containing the name of the [Font](#) to be used for rendering this item

**Returns:**

Nothing

**6.150.2.5 void CEGUI::ListboxTextItem::setTextColours (const ColourRect &cols)** [inline]

Set the colours used for text rendering.

**Parameters:**

*cols* [ColourRect](#) object describing the colours to be used.

**Returns:**

Nothing.

**6.150.2.6 void CEGUI::ListboxTextItem::setTextColours (colour top\_left\_colour, colour top\_right\_colour, colour bottom\_left\_colour, colour bottom\_right\_colour)**

Set the colours used for text rendering.

**Parameters:**

*top\_left\_colour* Colour (as ARGB value) to be applied to the top-left corner of each text glyph rendered.

*top\_right\_colour* Colour (as ARGB value) to be applied to the top-right corner of each text glyph rendered.

*bottom\_left\_colour* Colour (as ARGB value) to be applied to the bottom-left corner of each text glyph rendered.

*bottom\_right\_colour* Colour (as ARGB value) to be applied to the bottom-right corner of each text glyph rendered.

**Returns:**

Nothing.

**6.150.2.7 void CEGUI::ListboxTextItem::setTextColours (colour *col*) [inline]**

Set the colours used for text rendering.

**Parameters:**

*col* [colour](#) value to be used when rendering.

**Returns:**

Nothing.

**6.150.2.8 Size CEGUI::ListboxTextItem::getPixelSize (void) const [virtual]**

Return the rendered pixel size of this list box item.

**Returns:**

[Size](#) object describing the size of the list box item in pixels.

Implements [CEGUI::ListboxItem](#).

**6.150.2.9 void CEGUI::ListboxTextItem::draw (const Vector3 & *position*, float *alpha*, const Rect & *clipper*) const [virtual]**

Draw the list box item in its current state.

**Parameters:**

*position* Vecor3 object describing the upper-left corner of area that should be rendered in to for the draw operation.

*alpha* Alpha value to be used when rendering the item (between 0.0f and 1.0f).

*clipper* [Rect](#) object describing the clipping rectangle for the draw operation.

**Returns:**

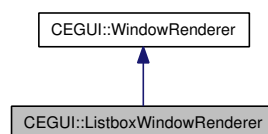
Nothing.

Implements [CEGUI::ListboxItem](#).

## 6.151 CEGUI::ListboxWindowRenderer Class Reference

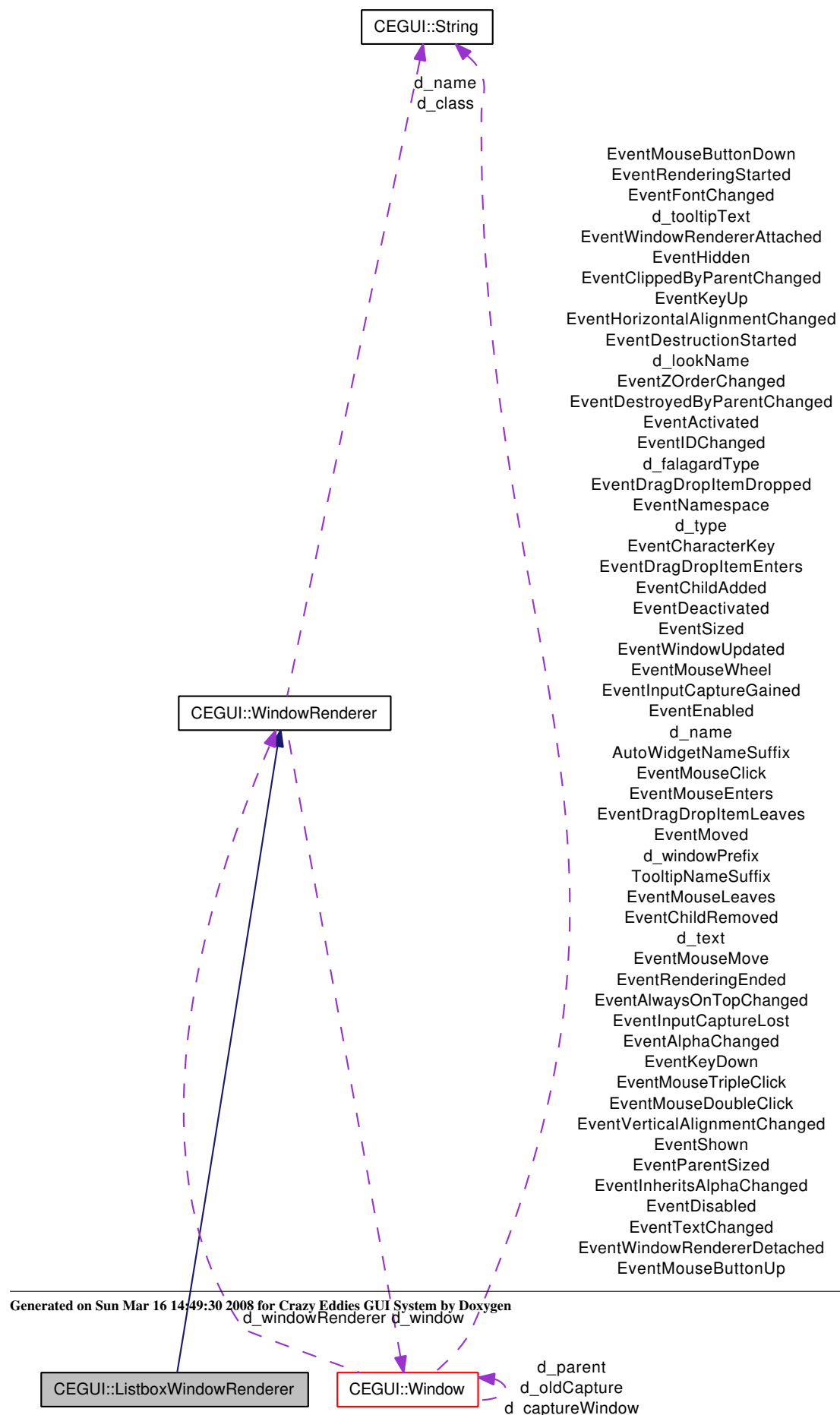
Base class for [Listbox](#) window renderer.

Inheritance diagram for CEGUI::ListboxWindowRenderer:





Collaboration diagram for CEGUI::ListboxWindowRenderer:



## Public Member Functions

- [ListboxWindowRenderer](#) (const [String](#) &name)

*Constructor.*

- virtual [Rect](#) [getListRenderArea](#) (void) const =0

*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*

### 6.151.1 Detailed Description

Base class for [Listbox](#) window renderer.

### 6.151.2 Member Function Documentation

#### 6.151.2.1 virtual [Rect](#) CEGUI::ListboxWindowRenderer::getListRenderArea (void) const [pure virtual]

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

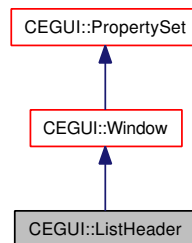
#### Returns:

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

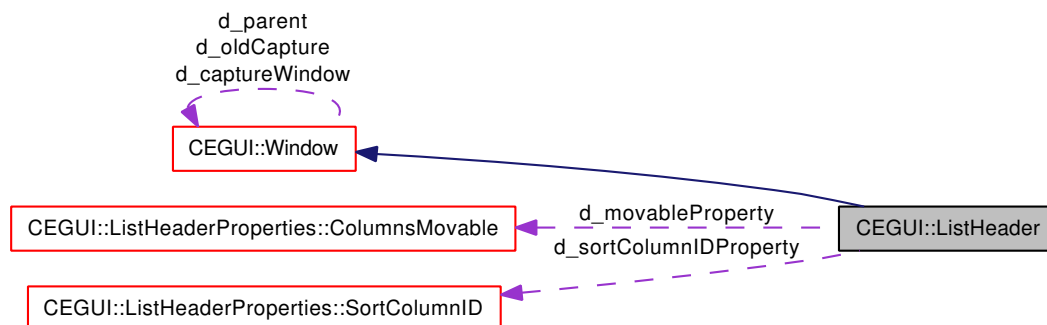
## 6.152 CEGUI::ListHeader Class Reference

Base class for the multi column list header widget.

Inheritance diagram for CEGUI::ListHeader:



Collaboration diagram for CEGUI::ListHeader:



### Public Member Functions

- `uint getColumnCount (void) const`  
Return the number of columns or segments attached to the header.
- `ListHeaderSegment & getSegmentFromColumn (uint column) const`  
Return the *ListHeaderSegment* object for the specified column.
- `ListHeaderSegment & getSegmentFromID (uint id) const`  
Return the *ListHeaderSegment* object with the specified ID.
- `ListHeaderSegment & getSortSegment (void) const`  
Return the *ListHeaderSegment* that is marked as being the 'sort key' segment. There must be at least one segment to successfully call this method.
- `uint getColumnFromSegment (const ListHeaderSegment &segment) const`  
Return the zero based column index of the specified segment.
- `uint getColumnFromID (uint id) const`  
Return the zero based column index of the segment with the specified ID.

- uint [getSortColumn](#) (void) const  
*Return the zero based index of the current sort column. There must be at least one segment/column to successfully call this method.*
- uint [getColumnWithText](#) (const [String](#) &text) const  
*Return the zero based column index of the segment with the specified text.*
- float [getPixelOffsetToSegment](#) (const [ListHeaderSegment](#) &segment) const  
*Return the pixel offset to the given [ListHeaderSegment](#).*
- float [getPixelOffsetToColumn](#) (uint column) const  
*Return the pixel offset to the [ListHeaderSegment](#) at the given zero based column index.*
- float [getTotalSegmentsPixelExtent](#) (void) const  
*Return the total pixel width of all attached segments.*
- [UDim](#) [getColumnWidth](#) (uint column) const  
*Return the width of the specified column.*
- [ListHeaderSegment::SortDirection](#) [getSortDirection](#) (void) const  
*Return the currently set sort direction.*
- bool [isSortingEnabled](#) (void) const  
*Return whether user manipulation of the sort column & direction are enabled.*
- bool [isColumnSizingEnabled](#) (void) const  
*Return whether the user may size column segments.*
- bool [isColumnDraggingEnabled](#) (void) const  
*Return whether the user may modify the order of the segments.*
- float [getSegmentOffset](#) (void) const  
*Return the current segment offset value. This value is used to implement scrolling of the header segments within the [ListHeader](#) area.*
- void [setSortingEnabled](#) (bool setting)  
*Set whether user manipulation of the sort column and direction is enabled.*
- void [setSortDirection](#) ([ListHeaderSegment::SortDirection](#) direction)  
*Set the current sort direction.*
- void [setSortSegment](#) (const [ListHeaderSegment](#) &segment)  
*Set the column segment to be used as the sort column.*
- void [setSortColumn](#) (uint column)  
*Set the column to be used as the sort column.*
- void [setSortColumnFromID](#) (uint id)  
*Set the column to to be used for sorting via its ID code.*

- void [setColumnSizingEnabled](#) (bool setting)  
*Set whether columns may be sized by the user.*
- void [setColumnDraggingEnabled](#) (bool setting)  
*Set whether columns may be reordered by the user via drag and drop.*
- void [addColumn](#) (const [String](#) &text, uint id, const [UDim](#) &width)  
*Add a new column segment to the end of the header.*
- void [insertColumn](#) (const [String](#) &text, uint id, const [UDim](#) &width, uint position)  
*Insert a new column segment at the specified position.*
- void [insertColumn](#) (const [String](#) &text, uint id, const [UDim](#) &width, const [ListHeaderSegment](#) &position)  
*Insert a new column segment at the specified position.*
- void [removeColumn](#) (uint column)  
*Removes a column segment from the [ListHeader](#).*
- void [removeSegment](#) (const [ListHeaderSegment](#) &segment)  
*Remove the specified segment from the [ListHeader](#).*
- void [moveColumn](#) (uint column, uint position)  
*Moves a column segment into a new position.*
- void [moveColumn](#) (uint column, const [ListHeaderSegment](#) &position)  
*Move a column segment to a new position.*
- void [moveSegment](#) (const [ListHeaderSegment](#) &segment, uint position)  
*Moves a segment into a new position.*
- void [moveSegment](#) (const [ListHeaderSegment](#) &segment, const [ListHeaderSegment](#) &position)  
*Move a segment to a new position.*
- void [setSegmentOffset](#) (float offset)  
*Set the current base segment offset. (This implements scrolling of the header segments within the header area).*
- void [setColumnWidth](#) (uint column, const [UDim](#) &width)  
*Set the width of the specified column.*
- [ListHeader](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for the list header base class.*
- virtual [~ListHeader](#) (void)  
*Destructor for the list header base class.*

## Static Public Attributes

- static const [String EventNamespace](#)  
*Namespace for global events.*
- static const [String WidgetTypeName](#)  
*Window factory name.*
- static const [String EventSortColumnChanged](#)  
*Event fired when the current sort column changes.*
- static const [String EventSortDirectionChanged](#)  
*Event fired when the sort direction changes.*
- static const [String EventSegmentSized](#)  
*Event fired when a segment has been sized by the user (e.window is the segment).*
- static const [String EventSegmentClicked](#)  
*Event fired when a segment has been clicked by the user (e.window is the segment).*
- static const [String EventSplitterDoubleClicked](#)  
*Event fired when a segment splitter has been double-clicked. (e.window is the segment).*
- static const [String EventSegmentSequenceChanged](#)  
*Event fired when the order of the segments has changed. ('e' is a [HeaderSequenceEventArgs](#)).*
- static const [String EventSegmentAdded](#)  
*Event fired when a segment is added to the header.*
- static const [String EventSegmentRemoved](#)  
*Event fired when a segment is removed from the header.*
- static const [String EventSortSettingChanged](#)  
*Event fired when setting that controls user modification to sort configuration changes.*
- static const [String EventDragMoveSettingChanged](#)  
*Event fired when setting that controls user drag & drop of segments changes.*
- static const [String EventDragSizeSettingChanged](#)  
*Event fired when setting that controls user sizing of segments changes.*
- static const [String EventSegmentRenderOffsetChanged](#)  
*Event fired when the rendering offset for the segments changes.*
- static const float [ScrollSpeed](#) = 8.0f  
*Speed to scroll at when dragging outside header.*
- static const float [MinimumSegmentPixelWidth](#) = 20.0f  
*Miniumum width of a segment in pixels.*

- static const char [SegmentNameSuffix](#) [] = "\_\_auto\_seg\_"  
*Widget name suffix for header segments.*

## Protected Types

- typedef std::vector< [ListHeaderSegment](#) \* > [SegmentList](#)

## Protected Member Functions

- [ListHeaderSegment](#) \* [createInitialisedSegment](#) (const [String](#) &text, uint id, const [UDim](#) &width)  
*Create and return a pointer to a new [ListHeaderSegment](#) based object.*
- void [layoutSegments](#) (void)  
*Layout the attached segments.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- [ListHeaderSegment](#) \* [createNewSegment](#) (const [String](#) &name) const  
*Create and return a pointer to a new [ListHeaderSegment](#) based object.*
- void [destroyListSegment](#) ([ListHeaderSegment](#) \*segment) const  
*Cleanup and destroy the given [ListHeaderSegment](#) that was created via the [createNewSegment](#) method.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- virtual void [onSortColumnChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the sort column is changed.*
- virtual void [onSortDirectionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the sort direction is changed.*
- virtual void [onSegmentSized](#) ([WindowEventArgs](#) &e)  
*Handler called when a segment is sized by the user. e.window points to the segment.*
- virtual void [onSegmentClicked](#) ([WindowEventArgs](#) &e)  
*Handler called when a segment is clicked by the user. e.window points to the segment.*
- virtual void [onSplitterDoubleClicked](#) ([WindowEventArgs](#) &e)  
*Handler called when a segment splitter / sizer is double-clicked. e.window points to the segment.*
- virtual void [onSegmentSequenceChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the segment / column order changes.*
- virtual void [onSegmentAdded](#) ([WindowEventArgs](#) &e)  
*Handler called when a new segment is added to the header.*

- virtual void [onSegmentRemoved](#) ([WindowEventArgs](#) &e)  
*Handler called when a segment is removed from the header.*
- virtual void [onSortSettingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called then setting that controls the users ability to modify the search column & direction changes.*
- virtual void [onDragMoveSettingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the setting that controls the users ability to drag and drop segments changes.*
- virtual void [onDragSizeSettingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the setting that controls the users ability to size segments changes.*
- virtual void [onSegmentOffsetChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the base rendering offset for the segments (scroll position) changes.*
- bool [segmentSizedHandler](#) (const [EventArgs](#) &e)
- bool [segmentMovedHandler](#) (const [EventArgs](#) &e)
- bool [segmentClickedHandler](#) (const [EventArgs](#) &e)
- bool [segmentDoubleClickHandler](#) (const [EventArgs](#) &e)
- bool [segmentDragHandler](#) (const [EventArgs](#) &e)

## Protected Attributes

- SegmentList [d\\_segments](#)  
*Attached segment windows in header order.*
- [ListHeaderSegment](#) \* [d\\_sortSegment](#)  
*Pointer to the segment that is currently set as the sort-key.*
- bool [d\\_sizingEnabled](#)  
*true if segments can be sized by the user.*
- bool [d\\_sortingEnabled](#)  
*true if the sort criteria modifications by user are enabled (no sorting is actually done)*
- bool [d\\_movingEnabled](#)  
*true if drag & drop moving of columns / segments is enabled.*
- uint [d\\_uniqueIDNumber](#)  
*field used to create unique names.*
- float [d\\_segmentOffset](#)  
*Base offset used to layout the segments (allows scrolling within the window area).*
- [ListHeaderSegment::SortDirection](#) [d\\_sortDir](#)  
*Brief copy of the current sort direction.*



### 6.152.1 Detailed Description

Base class for the multi column list header widget.

### 6.152.2 Member Function Documentation

#### 6.152.2.1 uint CEGUI::ListHeader::getColumnCount (void) const

Return the number of columns or segments attached to the header.

**Returns:**

uint value equal to the number of columns / segments currently in the header.

#### 6.152.2.2 ListHeaderSegment & CEGUI::ListHeader::getSegmentFromColumn (uint *column*) const

Return the [ListHeaderSegment](#) object for the specified column.

**Parameters:**

*column* zero based column index of the [ListHeaderSegment](#) to be returned.

**Returns:**

[ListHeaderSegment](#) object at the requested index.

**Exceptions:**

[InvalidRequestException](#) thrown if column is out of range.

#### 6.152.2.3 ListHeaderSegment & CEGUI::ListHeader::getSegmentFromID (uint *id*) const

Return the [ListHeaderSegment](#) object with the specified ID.

**Parameters:**

*id* id code of the [ListHeaderSegment](#) to be returned.

**Returns:**

[ListHeaderSegment](#) object with the ID *id*. If more than one segment has the same ID, only the first one will ever be returned.

**Exceptions:**

[InvalidRequestException](#) thrown if no segment with the requested ID is attached.

**6.152.2.4 ListHeaderSegment & CEGUI::ListHeader::getSortSegment (void) const**

Return the [ListHeaderSegment](#) that is marked as being the 'sort key' segment. There must be at least one segment to successfully call this method.

**Returns:**

[ListHeaderSegment](#) object which is the sort-key segment.

**Exceptions:**

[InvalidRequestException](#) thrown if no segments are attached to the [ListHeader](#).

**6.152.2.5 uint CEGUI::ListHeader::getColumnFromSegment (const ListHeaderSegment & segment) const**

Return the zero based column index of the specified segment.

**Parameters:**

*segment* [ListHeaderSegment](#) whos zero based index is to be returned.

**Returns:**

Zero based column index of the [ListHeaderSegment](#) *segment*.

**Exceptions:**

[InvalidRequestException](#) thrown if *segment* is not attached to this [ListHeader](#).

**6.152.2.6 uint CEGUI::ListHeader::getColumnFromID (uint id) const**

Return the zero based column index of the segment with the specified ID.

**Parameters:**

*id* ID code of the segment whos column index is to be returned.

**Returns:**

Zero based column index of the first [ListHeaderSegment](#) whos ID matches *id*.

**Exceptions:**

[InvalidRequestException](#) thrown if no attached segment has the requested ID.

**6.152.2.7 uint CEGUI::ListHeader::getSortColumn (void) const**

Return the zero based index of the current sort column. There must be at least one segment/column to successfully call this method.

**Returns:**

Zero based column index that is the current sort column.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if there are no segments / columns in this [ListHeader](#).

**6.152.2.8 uint CEGUI::ListHeader::getColumnWithText (const String & text) const**

Return the zero based column index of the segment with the specified text.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

**Returns:**

Zero based column index of the segment with the specified text.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if no attached segments have the requested text.

**6.152.2.9 float CEGUI::ListHeader::getPixelOffsetToSegment (const ListHeaderSegment & segment) const**

Return the pixel offset to the given [ListHeaderSegment](#).

**Parameters:**

*segment* [ListHeaderSegment](#) object that the offset to is to be returned.

**Returns:**

The number of pixels up-to the begining of the [ListHeaderSegment](#) described by *segment*.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *segment* is not attached to the [ListHeader](#).

**6.152.2.10 float CEGUI::ListHeader::getPixelOffsetToColumn (uint column) const**

Return the pixel offset to the [ListHeaderSegment](#) at the given zero based column index.

**Parameters:**

*column* Zero based column index of the [ListHeaderSegment](#) whos pixel offset it to be returned.

**Returns:**

The number of pixels up-to the begining of the [ListHeaderSegment](#) located at zero based column index *column*.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *column* is out of range.

**6.152.2.11 float CEGUI::ListHeader::getTotalSegmentsPixelExtent (void) const**

Return the total pixel width of all attached segments.

**Returns:**

Sum of the pixel widths of all attached [ListHeaderSegment](#) objects.

**6.152.2.12 UDim CEGUI::ListHeader::getColumnWidth (uint *column*) const**

Return the width of the specified column.

**Parameters:**

*column* Zero based column index of the segment whose width is to be returned.

**Returns:**

[UDim](#) describing the width of the [ListHeaderSegment](#) at the zero based column index specified by *column*.

**Exceptions:**

[InvalidRequestException](#) thrown if *column* is out of range.

**6.152.2.13 ListHeaderSegment::SortDirection CEGUI::ListHeader::getSortDirection (void) const**

Return the currently set sort direction.

**Returns:**

One of the [ListHeaderSegment::SortDirection](#) enumerated values specifying the current sort direction.

**6.152.2.14 bool CEGUI::ListHeader::isSortingEnabled (void) const**

Return whether user manipulation of the sort column & direction are enabled.

**Returns:**

true if the user may interactively modify the sort column and direction. false if the user may not modify the sort column and direction (these can still be set programmatically).

**6.152.2.15 bool CEGUI::ListHeader::isColumnSizingEnabled (void) const**

Return whether the user may size column segments.

**Returns:**

true if the user may interactively modify the width of column segments, false if they may not.

**6.152.2.16 bool CEGUI::ListHeader::isColumnDraggingEnabled (void) const**

Return whether the user may modify the order of the segments.

**Returns:**

true if the user may interactively modify the order of the column segments, false if they may not.

**6.152.2.17 float CEGUI::ListHeader::getSegmentOffset (void) const** [inline]

Return the current segment offset value. This value is used to implement scrolling of the header segments within the [ListHeader](#) area.

**Returns:**

float value specifying the current segment offset value in whatever metrics system is active for the [ListHeader](#).

**6.152.2.18 void CEGUI::ListHeader::setSortingEnabled (bool *setting*)**

Set whether user manipulation of the sort column and direction is enabled.

**Parameters:**

- setting* • true to allow interactive user manipulation of the sort column and direction.
- false to disallow interactive user manipulation of the sort column and direction.

**Returns:**

Nothing.

**6.152.2.19 void CEGUI::ListHeader::setSortDirection (ListHeaderSegment::SortDirection *direction*)**

Set the current sort direction.

**Parameters:**

*direction* One of the [ListHeaderSegment::SortDirection](#) enumerated values indicating the sort direction to be used.

**Returns:**

Nothing.

**6.152.2.20 void CEGUI::ListHeader::setSortSegment (const ListHeaderSegment & *segment*)**

Set the column segment to be used as the sort column.

**Parameters:**

*segment* [ListHeaderSegment](#) object indicating the column to be sorted.

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *segment* is not attached to this [ListHeader](#).

**6.152.2.21 void CEGUI::ListHeader::setSortColumn (uint *column*)**

Set the column to be used as the sort column.

**Parameters:**

*column* Zero based column index indicating the column to be sorted.

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *column* is out of range for this [ListHeader](#).

**6.152.2.22 void CEGUI::ListHeader::setSortColumnFromID (uint *id*)**

Set the column to be used for sorting via its ID code.

**Parameters:**

*id* ID code of the column segment that is to be used as the sort column.

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if no segment with ID *id* is attached to the [ListHeader](#).

**6.152.2.23 void CEGUI::ListHeader::setColumnSizingEnabled (bool *setting*)**

Set whether columns may be sized by the user.

**Parameters:**

- setting*
- true to indicate that the user may interactively size segments.
  - false to indicate that the user may not interactively size segments.

**Returns:**

Nothing.

**6.152.2.24 void CEGUI::ListHeader::setColumnDraggingEnabled (bool *setting*)**

Set whether columns may be reordered by the user via drag and drop.

**Parameters:**

- setting* • true to indicate the user may change the order of the column segments via drag and drop.
- false to indicate the user may not change the column segment order.

**Returns:**

Nothing.

**6.152.2.25 void CEGUI::ListHeader::addColumn (const String & *text*, uint *id*, const UDim & *width*)**

Add a new column segment to the end of the header.

**Parameters:**

- text* String object holding the initial text for the new segment
- id* Client specified ID code to be assigned to the new segment.
- width* UDim describing the initial width of the new segment.

**Returns:**

Nothing.

**6.152.2.26 void CEGUI::ListHeader::insertColumn (const String & *text*, uint *id*, const UDim & *width*, uint *position*)**

Insert a new column segment at the specified position.

**Parameters:**

- text* String object holding the initial text for the new segment
- id* Client specified ID code to be assigned to the new segment.
- width* UDim describing the initial width of the new segment.
- position* Zero based column index indicating the desired position for the new column. If this is greater than the current number of columns, the new segment is added to the end of the header.

**Returns:**

Nothing.

**6.152.2.27 void CEGUI::ListHeader::insertColumn (const String & *text*, uint *id*, const UDim & *width*, const ListHeaderSegment & *position*)**

Insert a new column segment at the specified position.

**Parameters:**

- text* [String](#) object holding the initial text for the new segment
- id* Client specified ID code to be assigned to the new segment.
- width* [UDim](#) describing the initial width of the new segment.
- position* [ListHeaderSegment](#) object indicating the insert position for the new segment. The new segment will be inserted before the segment indicated by *position*.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if [ListHeaderSegment](#) *position* is not attached to the [ListHeader](#).

**6.152.2.28 void CEGUI::ListHeader::removeColumn (uint *column*)**

Removes a column segment from the [ListHeader](#).

**Parameters:**

*column* Zero based column index indicating the segment to be removed.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *column* is out of range.

**6.152.2.29 void CEGUI::ListHeader::removeSegment (const [ListHeaderSegment](#) & *segment*)**

Remove the specified segment from the [ListHeader](#).

**Parameters:**

*segment* [ListHeaderSegment](#) object that is to be removed from the [ListHeader](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *segment* is not attached to this [ListHeader](#).



**6.152.2.30 void CEGUI::ListHeader::moveColumn (uint *column*, uint *position*)**

Moves a column segment into a new position.

**Parameters:**

*column* Zero based column index indicating the column segment to be moved.

*position* Zero based column index indicating the new position for the segment. If this is greater than the current number of segments, the segment is moved to the end of the header.

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *column* is out of range for this [ListHeader](#).

**6.152.2.31 void CEGUI::ListHeader::moveColumn (uint *column*, const ListHeaderSegment & *position*)**

Move a column segment to a new position.

**Parameters:**

*column* Zero based column index indicating the column segment to be moved.

*position* [ListHeaderSegment](#) object indicating the new position for the segment. The segment at *column* will be moved behind segment *position* (that is, segment *column* will appear to the right of segment *position*).

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *column* is out of range for this [ListHeader](#), or if *position* is not attached to this [ListHeader](#).

**6.152.2.32 void CEGUI::ListHeader::moveSegment (const ListHeaderSegment & *segment*, uint *position*)**

Moves a segment into a new position.

**Parameters:**

*segment* [ListHeaderSegment](#) object that is to be moved.

*position* Zero based column index indicating the new position for the segment. If this is greater than the current number of segments, the segment is moved to the end of the header.

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *segment* is not attached to this [ListHeader](#).

**6.152.2.33 void CEGUI::ListHeader::moveSegment (const ListHeaderSegment & *segment*, const ListHeaderSegment & *position*)**

Move a segment to a new position.

**Parameters:**

*segment* [ListHeaderSegment](#) object that is to be moved.

*position* [ListHeaderSegment](#) object indicating the new position for the segment. The segment *segment* will be moved behind segment *position* (that is, segment *segment* will appear to the right of segment *position*).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if either *segment* or *position* are not attached to this [ListHeader](#).

**6.152.2.34 void CEGUI::ListHeader::setSegmentOffset (float *offset*)**

Set the current base segment offset. (This implements scrolling of the header segments within the header area).

**Parameters:**

*offset* New base offset for the first segment. The segments will be offset to the left by the amount specified. *offset* should be specified using the active metrics system for the [ListHeader](#).

**Returns:**

Nothing.

**6.152.2.35 void CEGUI::ListHeader::setColumnWidth (uint *column*, const UDim & *width*)**

Set the width of the specified column.

**Parameters:**

*column* Zero based column index of the segment whose width is to be set.

*width* [UDim](#) value specifying the new width to set for the [ListHeaderSegment](#) at the zero based column index specified by *column*.

**Returns:**

Nothing

**Exceptions:**

[InvalidRequestException](#) thrown if *column* is out of range.

**6.152.2.36** `ListHeaderSegment * CEGUI::ListHeader::createInitialisedSegment (const String & text, uint id, const UDim & width)` [protected]

Create and return a pointer to a new [ListHeaderSegment](#) based object.

**Parameters:**

*name* [String](#) object holding the name that should be given to the new [Window](#).

**Returns:**

Pointer to an [ListHeaderSegment](#) based object of whatever type is appropriate for this [ListHeader](#).

Cleanup and destroy the given [ListHeaderSegment](#) that was created via the createNewSegment method.

**Parameters:**

*segment* Pointer to a [ListHeaderSegment](#) based object to be destroyed.

**Returns:**

Nothing.

Create initialise and return a [ListHeaderSegment](#) object, with all events subscribed and ready to use.

**6.152.2.37** `virtual bool CEGUI::ListHeader::testClassName_impl (const String & class_name)`  
`const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.152.2.38** `ListHeaderSegment * CEGUI::ListHeader::createNewSegment (const String & name)`  
`const` [protected]

Create and return a pointer to a new [ListHeaderSegment](#) based object.

**Parameters:**

*name* [String](#) object holding the name that should be given to the new [Window](#).

**Returns:**

Pointer to an [ListHeaderSegment](#) based object of whatever type is appropriate for this [ListHeader](#).

**6.152.2.39** `void CEGUI::ListHeader::destroyListSegment (ListHeaderSegment * segment) const`  
[protected]

Cleanup and destroy the given [ListHeaderSegment](#) that was created via the `createNewSegment` method.

**Parameters:**

*segment* Pointer to a [ListHeaderSegment](#) based object to be destroyed.

**Returns:**

Nothing.

**6.152.2.40** `virtual bool CEGUI::ListHeader::validateWindowRenderer (const String & name)`  
`const` [inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

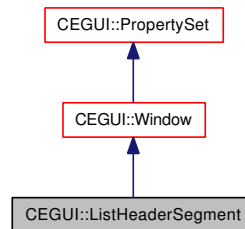
Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

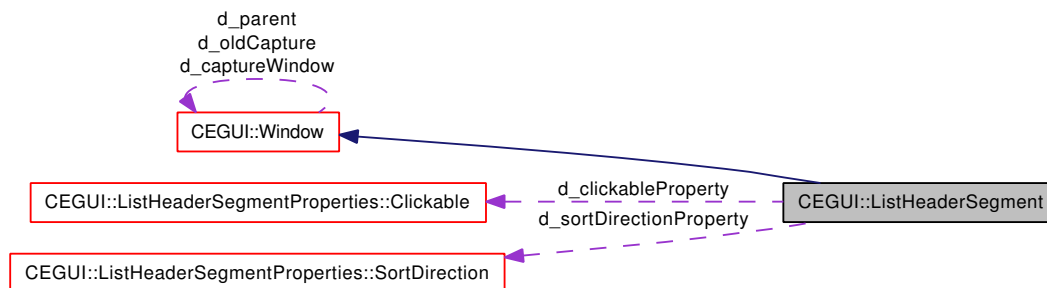
## 6.153 CEGUI::ListHeaderSegment Class Reference

Base class for list header segment window.

Inheritance diagram for CEGUI::ListHeaderSegment:



Collaboration diagram for CEGUI::ListHeaderSegment:



### Public Types

- enum `SortDirection` { `None`, `Ascending`, `Descending` }  
*Enumeration of possible values for sorting direction used with [ListHeaderSegment](#) classes.*

### Public Member Functions

- bool `isSizingEnabled` (void) const  
*Return whether this segment can be sized.*
- `SortDirection` `getSortDirection` (void) const  
*Return the current sort direction set for this segment.*
- bool `isDragMovingEnabled` (void) const  
*Return whether drag moving is enabled for this segment.*
- const `Point` & `getDragMoveOffset` (void) const  
*Return the current drag move position offset (in pixels relative to the top-left corner of the segment).*
- bool `isClickable` (void) const  
*Return whether the segment is clickable.*

- bool [isSegmentHovering](#) (void) const  
*Return whether the segment is currently in its hovering state.*
- bool [isSegmentPushed](#) (void) const  
*Return whether the segment is currently in its pushed state.*
- bool [isSplitterHovering](#) (void) const  
*Return whether the splitter is currently in its hovering state.*
- bool [isBeingDragMoved](#) (void) const  
*Return whether the segment is currently being drag-moved.*
- bool [isBeingDragSized](#) (void) const  
*Return whether the segment is currently being drag-moved.*
- const [Image](#) \* [getSizingCursorImage](#) () const
- const [Image](#) \* [getMovingCursorImage](#) () const
- void [setSizingEnabled](#) (bool setting)  
*Set whether this segment can be sized.*
- void [setSortDirection](#) ([SortDirection](#) sort\_dir)  
*Set the current sort direction set for this segment.*
- void [setDragMovingEnabled](#) (bool setting)  
*Set whether drag moving is allowed for this segment.*
- void [setClickable](#) (bool setting)  
*Set whether the segment is clickable.*
- void [setSizingCursorImage](#) (const [Image](#) \*image)
- void [setSizingCursorImage](#) (const [String](#) &imageset, const [String](#) &image)
- void [setMovingCursorImage](#) (const [Image](#) \*image)
- void [setMovingCursorImage](#) (const [String](#) &imageset, const [String](#) &image)
- [ListHeaderSegment](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for list header segment base class.*
- virtual [~ListHeaderSegment](#) (void)  
*Destructor for list header segment base class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*Window factory name.*

- static const [String EventSegmentClicked](#)  
*Event fired when the segment is clicked.*
- static const [String EventSplitterDoubleClicked](#)  
*Event fired when the sizer/splitter is double-clicked.*
- static const [String EventSizingSettingChanged](#)  
*Event fired when the sizing setting changes.*
- static const [String EventSortDirectionChanged](#)  
*Event fired when the sort direction value changes.*
- static const [String EventMovableSettingChanged](#)  
*Event fired when the movable setting changes.*
- static const [String EventSegmentDragStart](#)  
*Event fired when the segment has started to be dragged.*
- static const [String EventSegmentDragStop](#)  
*Event fired when segment dragging has stopped (via mouse release).*
- static const [String EventSegmentDragPositionChanged](#)  
*Event fired when the segment drag position has changed.*
- static const [String EventSegmentSized](#)  
*Event fired when the segment is sized.*
- static const [String EventClickableSettingChanged](#)  
*Event fired when the clickable state of the segment changes.*
- static const float [DefaultSizingArea](#) = 8.0f  
*Default size of the sizing area.*
- static const float [SegmentMoveThreshold](#) = 12.0f  
*Amount the mouse must be dragged before drag-moving is initiated.*

## Protected Member Functions

- void [doDragSizing](#) (const [Point](#) &local\_mouse)  
*Update state for drag sizing.*
- void [doDragMoving](#) (const [Point](#) &local\_mouse)  
*Update state for drag moving.*
- void [initDragMoving](#) (void)  
*Initialise the required states to put the widget into drag-moving mode.*
- void [initSizingHoverState](#) (void)

*Initialise the required states when we are hovering over the sizing area.*

- void [initSegmentHoverState](#) (void)  
*Initialise the required states when we are hovering over the main segment area.*
- bool [isDragMoveThresholdExceeded](#) (const [Point](#) &local\_mouse)  
*Return whether the required minimum movement threshold before initiating drag-moving has been exceeded.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [onSegmentClicked](#) ([WindowEventArgs](#) &e)  
*Handler called when segment is clicked.*
- virtual void [onSplitterDoubleClicked](#) ([WindowEventArgs](#) &e)  
*Handler called when the sizer/splitter is double-clicked.*
- virtual void [onSizingSettingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when sizing setting changes.*
- virtual void [onSortDirectionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the sort direction value changes.*
- virtual void [onMovableSettingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the drag-movable setting is changed.*
- virtual void [onSegmentDragStart](#) ([WindowEventArgs](#) &e)  
*Handler called when the user starts dragging the segment.*
- virtual void [onSegmentDragStop](#) ([WindowEventArgs](#) &e)  
*Handler called when the user stops dragging the segment (releases mouse button).*
- virtual void [onSegmentDragPositionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the drag position changes.*
- virtual void [onSegmentSized](#) ([WindowEventArgs](#) &e)  
*Handler called when the segment is sized.*
- virtual void [onClickableViewChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the clickable setting for the segment changes.*
- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onMouseDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*



- virtual void [onMouseDoubleClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been double-clicked within this window's area.*
- virtual void [onMouseLeaves](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has left this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*

## Protected Attributes

- const [Image](#) \* [d\\_sizingMouseCursor](#)  
*Image to use for mouse when sizing (typically set by derived class).*
- const [Image](#) \* [d\\_movingMouseCursor](#)  
*Image to use for mouse when moving (typically set by derived class).*
- float [d\\_splitterSize](#)  
*pixel width of the sizing area.*
- bool [d\\_splitterHover](#)  
*True if the mouse is over the splitter.*
- bool [d\\_dragSizing](#)  
*true when we are being sized.*
- [Point](#) [d\\_dragPoint](#)  
*point we are being dragged at when sizing or moving.*
- [SortDirection](#) [d\\_sortDir](#)  
*Direction for sorting (used for deciding what icon to display).*
- bool [d\\_segmentHover](#)  
*true when the mouse is within the segment area (and not in sizing area).*
- bool [d\\_segmentPushed](#)  
*true when the left mouse button has been pressed within the confines of the segment.*
- bool [d\\_sizingEnabled](#)  
*true when sizing is enabled for this segment.*
- bool [d\\_movingEnabled](#)  
*True when drag-moving is enabled for this segment;.*
- bool [d\\_dragMoving](#)  
*true when segment is being drag moved.*

- [Point d\\_dragPosition](#)  
*position of dragged segment.*
- [bool d\\_allowClicks](#)  
*true if the segment can be clicked.*

### 6.153.1 Detailed Description

Base class for list header segment window.

### 6.153.2 Member Enumeration Documentation

#### 6.153.2.1 `enum CEGUI::ListHeaderSegment::SortDirection`

Enumeration of possible values for sorting direction used with [ListHeaderSegment](#) classes.

**Enumerator:**

- None* Items under this segment should not be sorted.
- Ascending* Items under this segment should be sorted in ascending order.
- Descending* Items under this segment should be sorted in descending order.

### 6.153.3 Member Function Documentation

#### 6.153.3.1 `bool CEGUI::ListHeaderSegment::isSizingEnabled (void) const` `[inline]`

Return whether this segment can be sized.

**Returns:**

true if the segment can be horizontally sized, false if the segment can not be horizontally sized.

#### 6.153.3.2 `SortDirection CEGUI::ListHeaderSegment::getSortDirection (void) const` `[inline]`

Return the current sort direction set for this segment.

Note that this has no impact on the way the segment functions (aside from the possibility of varied rendering). This is intended as a 'helper setting' to classes that make use of the [ListHeaderSegment](#) objects.

**Returns:**

One of the SortDirection enumerated values indicating the current sort direction set for this segment.

#### 6.153.3.3 `bool CEGUI::ListHeaderSegment::isDragMovingEnabled (void) const` `[inline]`

Return whether drag moving is enabled for this segment.

**Returns:**

true if the segment can be drag moved, false if the segment can not be drag moved.

**6.153.3.4 const Point& CEGUI::ListHeaderSegment::getDragMoveOffset (void) const**  
[inline]

Return the current drag move position offset (in pixels relative to the top-left corner of the segment).

**Returns:**

Point object describing the drag move offset position.

**6.153.3.5 bool CEGUI::ListHeaderSegment::isClickable (void) const** [inline]

Return whether the segment is clickable.

**Returns:**

true if the segment can be clicked, false if the segment can not be clicked (so no highlighting or events will happen).

**6.153.3.6 void CEGUI::ListHeaderSegment::setSizeEnabled (bool *setting*)**

Set whether this segment can be sized.

**Parameters:**

*setting* true if the segment may be horizontally sized, false if the segment may not be horizontally sized.

**Returns:**

Nothing.

**6.153.3.7 void CEGUI::ListHeaderSegment::setSortDirection (SortDirection *sort\_dir*)**

Set the current sort direction set for this segment.

Note that this has no impact on the way the segment functions (aside from the possibility of varied rendering). This is intended as a 'helper setting' to classes that make use of the [ListHeaderSegment](#) objects.

**Parameters:**

*sort\_dir* One of the SortDirection enumerated values indicating the current sort direction set for this segment.

**Returns:**

Nothing

**6.153.3.8 void CEGUI::ListHeaderSegment::setDragMovingEnabled (bool *setting*)**

Set whether drag moving is allowed for this segment.

**Parameters:**

*setting* true if the segment may be drag moved, false if the segment may not be drag moved.

**Returns:**

Nothing.

**6.153.3.9 void CEGUI::ListHeaderSegment::setClickable (bool *setting*)**

Set whether the segment is clickable.

**Parameters:**

*setting* true if the segment may be clicked, false if the segment may not be clicked (so no highlighting or events will happen).

**Returns:**

Nothing.

**6.153.3.10 void CEGUI::ListHeaderSegment::doDragSizing (const Point & *local\_mouse*)**  
[protected]

Update state for drag sizing.

**Parameters:**

*local\_mouse* Mouse position as a pixel offset from the top-left corner of this window.

**Returns:**

Nothing.

**6.153.3.11 void CEGUI::ListHeaderSegment::doDragMoving (const Point & *local\_mouse*)**  
[protected]

Update state for drag moving.

**Parameters:**

*local\_mouse* Mouse position as a pixel offset from the top-left corner of this window.

**Returns:**

Nothing.

**6.153.3.12** `bool CEGUI::ListHeaderSegment::isDragMoveThresholdExceeded (const Point & local_mouse)` [protected]

Return whether the required minimum movement threshold before initiating drag-moving has been exceeded.

**Parameters:**

*local\_mouse* Mouse position as a pixel offset from the top-left corner of this window.

**Returns:**

true if the threshold has been exceeded and drag-moving should be initiated, or false if the threshold has not been exceeded.

**6.153.3.13** `virtual bool CEGUI::ListHeaderSegment::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.153.3.14** `void CEGUI::ListHeaderSegment::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.153.3.15** `void CEGUI::ListHeaderSegment::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.153.3.16** `void CEGUI::ListHeaderSegment::onMouseButtonUp (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.153.3.17** `void CEGUI::ListHeaderSegment::onMouseDoubleClicked (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been double-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.153.3.18** `void CEGUI::ListHeaderSegment::onMouseLeaves (MouseEventArgs & e)`  
[protected, virtual]

Handler called when the mouse cursor has left this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.153.3.19** `void CEGUI::ListHeaderSegment::onCaptureLost (WindowEventArgs & e)`  
[protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

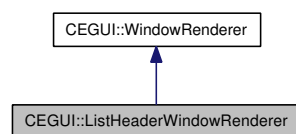
*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

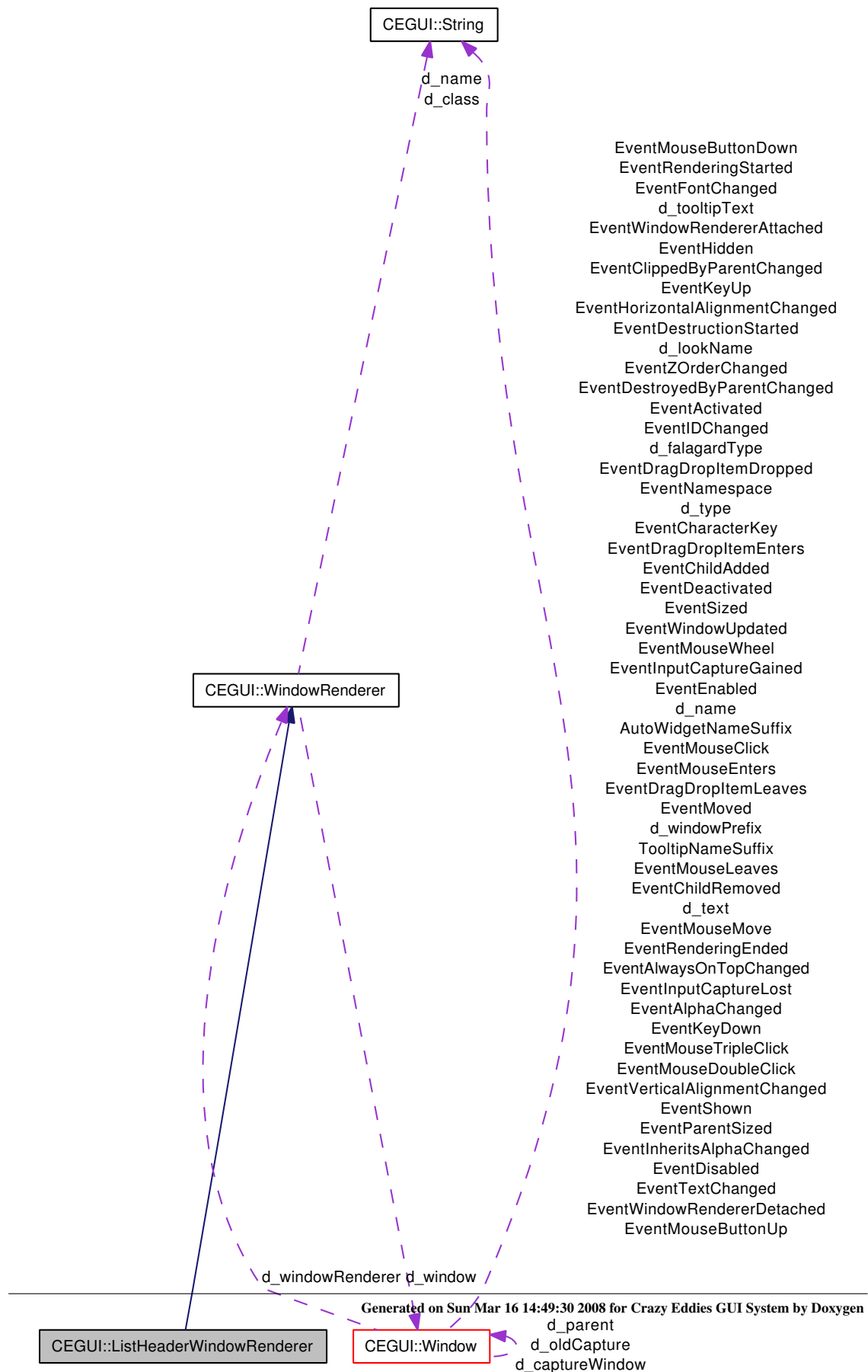
## 6.154 CEGUI::ListHeaderWindowRenderer Class Reference

Base class for the multi column list header window renderer.

Inheritance diagram for CEGUI::ListHeaderWindowRenderer:



Collaboration diagram for CEGUI::ListHeaderWindowRenderer:





## Public Member Functions

- [ListHeaderWindowRenderer](#) (const [String](#) &name)  
*Constructor.*
- virtual [ListHeaderSegment](#) \* [createNewSegment](#) (const [String](#) &name) const =0  
*Create and return a pointer to a new [ListHeaderSegment](#) based object.*
- virtual void [destroyListSegment](#) ([ListHeaderSegment](#) \*segment) const =0  
*Cleanup and destroy the given [ListHeaderSegment](#) that was created via the [createNewSegment](#) method.*

### 6.154.1 Detailed Description

Base class for the multi column list header window renderer.

### 6.154.2 Member Function Documentation

#### 6.154.2.1 virtual [ListHeaderSegment](#)\* CEGUI::ListHeaderWindowRenderer::createNewSegment (const [String](#) & name) const [pure virtual]

Create and return a pointer to a new [ListHeaderSegment](#) based object.

##### Parameters:

*name* [String](#) object holding the name that should be given to the new [Window](#).

##### Returns:

Pointer to an [ListHeaderSegment](#) based object of whatever type is appropriate for this [ListHeader](#).

#### 6.154.2.2 virtual void CEGUI::ListHeaderWindowRenderer::destroyListSegment ([ListHeaderSegment](#) \* segment) const [pure virtual]

Cleanup and destroy the given [ListHeaderSegment](#) that was created via the [createNewSegment](#) method.

##### Parameters:

*segment* Pointer to a [ListHeaderSegment](#) based object to be destroyed.

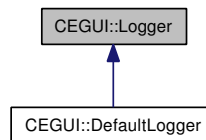
##### Returns:

Nothing.

## 6.155 CEGUI::Logger Class Reference

Abstract class that defines the interface of a logger object for the GUI system. The default implementation of this interface is the [DefaultLogger](#) class; if you want to perform special logging, derive your own class from [Logger](#) and initialize a object of that type before you create the [CEGUI::System](#) singleton.

Inheritance diagram for CEGUI::Logger:



### Public Member Functions

- [Logger](#) (void)  
*Constructor for [Logger](#) object.*
- virtual [~Logger](#) (void)  
*Destructor for [Logger](#) object.*
- void [setLoggingLevel](#) ([LoggingLevel](#) level)  
*Set the level of logging information that will get out to the log file.*
- [LoggingLevel](#) [getLoggingLevel](#) (void) const  
*return the current logging level setting*
- virtual void [logEvent](#) (const [String](#) &message, [LoggingLevel](#) level=Standard)=0  
*Add an event to the log.*
- virtual void [setLogFilename](#) (const [String](#) &filename, bool append=false)=0  
*Set the name of the log file where all subsequent log entries should be written. The interpretation of file name may differ depending on the concrete logger implementation.*

### Protected Attributes

- [LoggingLevel](#) d\_level  
*Holds current logging level.*

#### 6.155.1 Detailed Description

Abstract class that defines the interface of a logger object for the GUI system. The default implementation of this interface is the [DefaultLogger](#) class; if you want to perform special logging, derive your own class from [Logger](#) and initialize a object of that type before you create the [CEGUI::System](#) singleton.

## 6.155.2 Member Function Documentation

### 6.155.2.1 void CEGUI::Logger::setLoggingLevel (LoggingLevel *level*) [inline]

Set the level of logging information that will get out to the log file.

**Parameters:**

*level* One of the LoggingLevel enumerated values that specified the level of logging information required.

**Returns:**

Nothing

### 6.155.2.2 LoggingLevel CEGUI::Logger::getLoggingLevel (void) const [inline]

return the current logging level setting

**Returns:**

One of the LoggingLevel enumerated values specifying the current level of logging

### 6.155.2.3 virtual void CEGUI::Logger::logEvent (const String & *message*, LoggingLevel *level* = Standard) [pure virtual]

Add an event to the log.

**Parameters:**

*message* [String](#) object containing the message to be added to the event log.

*level* LoggingLevel for this message. If *level* is greater than the current set logging level, the message is not logged.

**Returns:**

Nothing

Implemented in [CEGUI::DefaultLogger](#).

### 6.155.2.4 virtual void CEGUI::Logger::setLogFilename (const String & *filename*, bool *append* = false) [pure virtual]

Set the name of the log file where all subsequent log entries should be written. The interpretation of file name may differ depending on the concrete logger implementation.

**Note:**

When this is called, and the log file is created, any cached log entries are flushed to the log file.

**Parameters:**

*filename* Name of the file to put log messages.

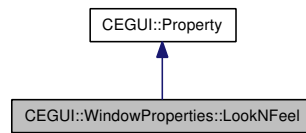
- append*
- true if events should be added to the end of the current file.
  - false if the current contents of the file should be discarded.

Implemented in [CEGUI::DefaultLogger](#).

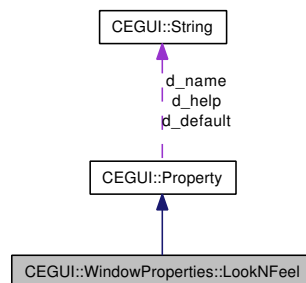
## 6.156 CEGUI::WindowProperties::LookNFeel Class Reference

[Property](#) to access/change the assigned look'n'feel.

Inheritance diagram for CEGUI::WindowProperties::LookNFeel:



Collaboration diagram for CEGUI::WindowProperties::LookNFeel:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- void [writeXMLToStream](#) (const [PropertyReceiver](#) \*receiver, [XMLSerializer](#) &xml\_stream) const  
*Writes out an XML representation of this class to the given stream.*

### 6.156.1 Detailed Description

[Property](#) to access/change the assigned look'n'feel.

Usage:

- Name: [LookNFeel](#)
- Format: "[LookNFeelName]"

Where [LookNFeelName] is the name of the look'n'feel you wish to assign.

## 6.156.2 Member Function Documentation

### 6.156.2.1 **String CEGUI::WindowProperties::LookNFeel::get (const PropertyReceiver \* *receiver*) const** [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.156.2.2 **void CEGUI::WindowProperties::LookNFeel::set (PropertyReceiver \* *receiver*, const String & *value*)** [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

### 6.156.2.3 **void CEGUI::WindowProperties::LookNFeel::writeXMLToStream (const PropertyReceiver \* *receiver*, XMLSerializer & *xml\_stream*) const** [virtual]

Writes out an XML representation of this class to the given stream.

#### Note:

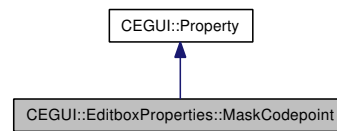
This would normally have been implemented via XMLGenerator base class, but in this case we require the target [PropertyReceiver](#) in order to obtain the property value.

Reimplemented from [CEGUI::Property](#).

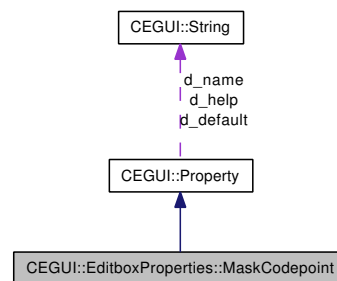
## 6.157 CEGUI::EditboxProperties::MaskCodepoint Class Reference

[Property](#) to access the mask text setting of the edit box.

Inheritance diagram for CEGUI::EditboxProperties::MaskCodepoint:



Collaboration diagram for CEGUI::EditboxProperties::MaskCodepoint:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.157.1 Detailed Description

[Property](#) to access the mask text setting of the edit box.

This property offers access to the mask text setting for the [Editbox](#) object.

##### Usage:

- Name: [MaskCodepoint](#)
- Format: "[uint]"

##### Where:

- [uint] is the Unicode utf32 value of the codepoint used for masking text.

## 6.157.2 Member Function Documentation

### 6.157.2.1 `String CEGUI::EditboxProperties::MaskCodepoint::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.157.2.2 `void CEGUI::EditboxProperties::MaskCodepoint::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

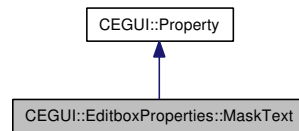
Implements [CEGUI::Property](#).



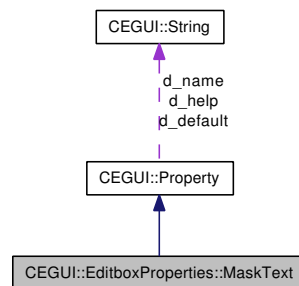
## 6.158 CEGUI::EditboxProperties::MaskText Class Reference

[Property](#) to access the mask text setting of the edit box.

Inheritance diagram for CEGUI::EditboxProperties::MaskText:



Collaboration diagram for CEGUI::EditboxProperties::MaskText:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.158.1 Detailed Description

[Property](#) to access the mask text setting of the edit box.

This property offers access to the mask text setting for the [Editbox](#) object.

**Usage:**

- Name: [MaskText](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate the text should be masked.
- "False" to indicate the text should not be masked.

## 6.158.2 Member Function Documentation

### 6.158.2.1 `String CEGUI::EditboxProperties::MaskText::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.158.2.2 `void CEGUI::EditboxProperties::MaskText::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

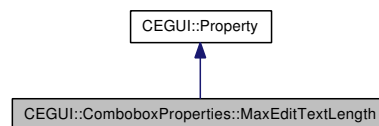
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

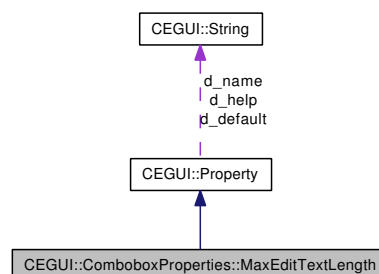
## 6.159 CEGUI::ComboboxProperties::MaxEditTextLength Class Reference

[Property](#) to access the maximum text length for the edit box.

Inheritance diagram for CEGUI::ComboboxProperties::MaxEditTextLength:



Collaboration diagram for CEGUI::ComboboxProperties::MaxEditTextLength:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.159.1 Detailed Description

[Property](#) to access the maximum text length for the edit box.

##### Usage:

- Name: [MaxEditTextLength](#)
- Format: "[uint]"

##### Where:

- [uint] is the maximum allowed text length (as a count of code points).

## 6.159.2 Member Function Documentation

### 6.159.2.1 String CEGUI::ComboboxProperties::MaxEditTextLength::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.159.2.2 void CEGUI::ComboboxProperties::MaxEditTextLength::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

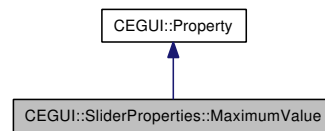
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

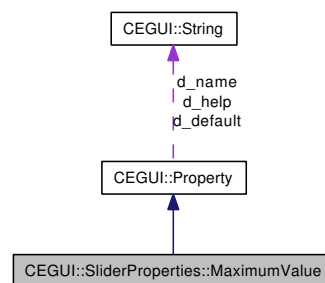
## 6.160 CEGUI::SliderProperties::MaximumValue Class Reference

[Property](#) to access the maximum value of the slider.

Inheritance diagram for CEGUI::SliderProperties::MaximumValue:



Collaboration diagram for CEGUI::SliderProperties::MaximumValue:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.160.1 Detailed Description

[Property](#) to access the maximum value of the slider.

#### Usage:

- Name: [MaximumValue](#)
- Format: "[float]".

#### Where:

- [float] represents the maximum value of the slider.

## 6.160.2 Member Function Documentation

### 6.160.2.1 `String CEGUI::SliderProperties::MaximumValue::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.160.2.2 `void CEGUI::SliderProperties::MaximumValue::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

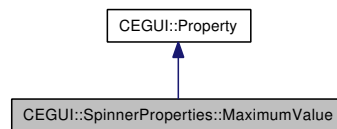
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

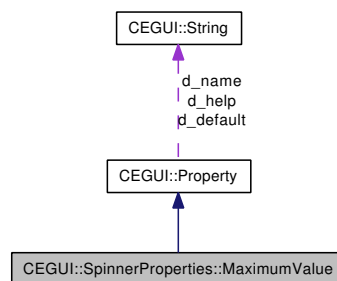
## 6.161 CEGUI::SpinnerProperties::MaximumValue Class Reference

[Property](#) to access the maximum value setting of the spinner.

Inheritance diagram for CEGUI::SpinnerProperties::MaximumValue:



Collaboration diagram for CEGUI::SpinnerProperties::MaximumValue:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.161.1 Detailed Description

[Property](#) to access the maximum value setting of the spinner.

Usage:

- Name: [MaximumValue](#)
- Format: "[float]".

Where:

- [float] represents the current maximum value of the [Spinner](#) widget.

## 6.161.2 Member Function Documentation

### 6.161.2.1 `String CEGUI::SpinnerProperties::MaximumValue::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.161.2.2 `void CEGUI::SpinnerProperties::MaximumValue::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

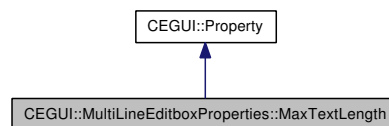
Implements [CEGUI::Property](#).



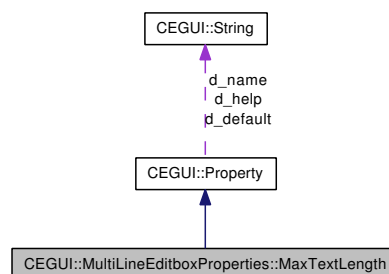
## 6.162 CEGUI::MultiLineEditboxProperties::MaxTextLength Class Reference

[Property](#) to access the maximum text length for the edit box.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::MaxTextLength:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::MaxTextLength:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.162.1 Detailed Description

[Property](#) to access the maximum text length for the edit box.

#### Usage:

- Name: [MaxTextLength](#)
- Format: "[uint]"

#### Where:

- [uint] is the maximum allowed text length (as a count of code points).

## 6.162.2 Member Function Documentation

### 6.162.2.1 String CEGUI::MultiLineEditboxProperties::MaxTextLength::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.162.2.2 void CEGUI::MultiLineEditboxProperties::MaxTextLength::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

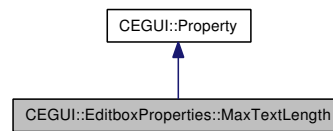
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

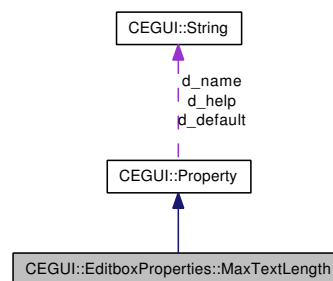
## 6.163 CEGUI::EditboxProperties::MaxTextLength Class Reference

[Property](#) to access the maximum text length for the edit box.

Inheritance diagram for CEGUI::EditboxProperties::MaxTextLength:



Collaboration diagram for CEGUI::EditboxProperties::MaxTextLength:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.163.1 Detailed Description

[Property](#) to access the maximum text length for the edit box.

#### Usage:

- Name: [MaxTextLength](#)
- Format: "[uint]"

#### Where:

- [uint] is the maximum allowed text length (as a count of code points).

## 6.163.2 Member Function Documentation

### 6.163.2.1 String CEGUI::EditboxProperties::MaxTextLength::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.163.2.2 void CEGUI::EditboxProperties::MaxTextLength::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.164 CEGUI::MCLGridRef Struct Reference

Simple grid index structure.

### Public Member Functions

- **MCLGridRef** (uint r, uint c)
- **MCLGridRef** & **operator=** (const **MCLGridRef** &rhs)
- bool **operator<** (const **MCLGridRef** &rhs) const
- bool **operator<=** (const **MCLGridRef** &rhs) const
- bool **operator>** (const **MCLGridRef** &rhs) const
- bool **operator>=** (const **MCLGridRef** &rhs) const
- bool **operator==** (const **MCLGridRef** &rhs) const
- bool **operator!=** (const **MCLGridRef** &rhs) const

### Public Attributes

- uint **row**  
*Zero based row index.*
- uint **column**  
*Zero based column index.*

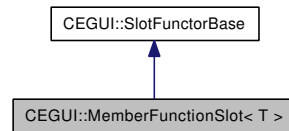
#### 6.164.1 Detailed Description

Simple grid index structure.

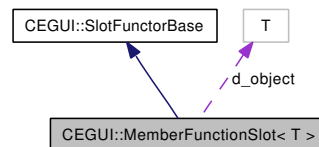
## 6.165 CEGUI::MemberFunctionSlot< T > Class Template Reference

Slot template class that creates a functor that calls back via a class member function.

Inheritance diagram for CEGUI::MemberFunctionSlot< T >:



Collaboration diagram for CEGUI::MemberFunctionSlot< T >:



### Public Types

- typedef bool(T::\* [MemberFunctionType](#))(const [EventArgs](#) &)  
*Member function slot type.*

### Public Member Functions

- **MemberFunctionSlot** ([MemberFunctionType](#) func, T \*obj)
- virtual bool **operator()** (const [EventArgs](#) &args)

#### 6.165.1 Detailed Description

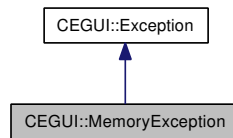
**template<typename T> class CEGUI::MemberFunctionSlot< T >**

Slot template class that creates a functor that calls back via a class member function.

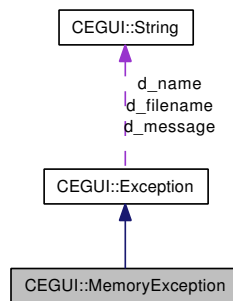
## 6.166 CEGUI::MemoryException Class Reference

[Exception](#) class used when a memory handling error is detected.

Inheritance diagram for CEGUI::MemoryException:



Collaboration diagram for CEGUI::MemoryException:



### Public Member Functions

- [MemoryException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the memory exception by calling the base class.*

### 6.166.1 Detailed Description

[Exception](#) class used when a memory handling error is detected.

### 6.166.2 Constructor & Destructor Documentation

#### 6.166.2.1 CEGUI::MemoryException::MemoryException (const [String](#) & message, const [String](#) & file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the memory exception by calling the base class.

#### Parameters:

- message* [String](#) object describing the reason for the memory exception being thrown.
- filename* [String](#) object containing the name of the file where the memory exception occurred.
- line* Integer representing the line number where the memory exception occurred.

**Remarks:**

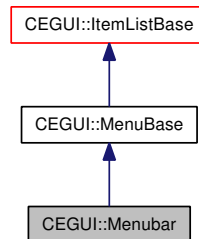
The memory exception name is automatically passed to the base class as "CEGUI::MemoryException".



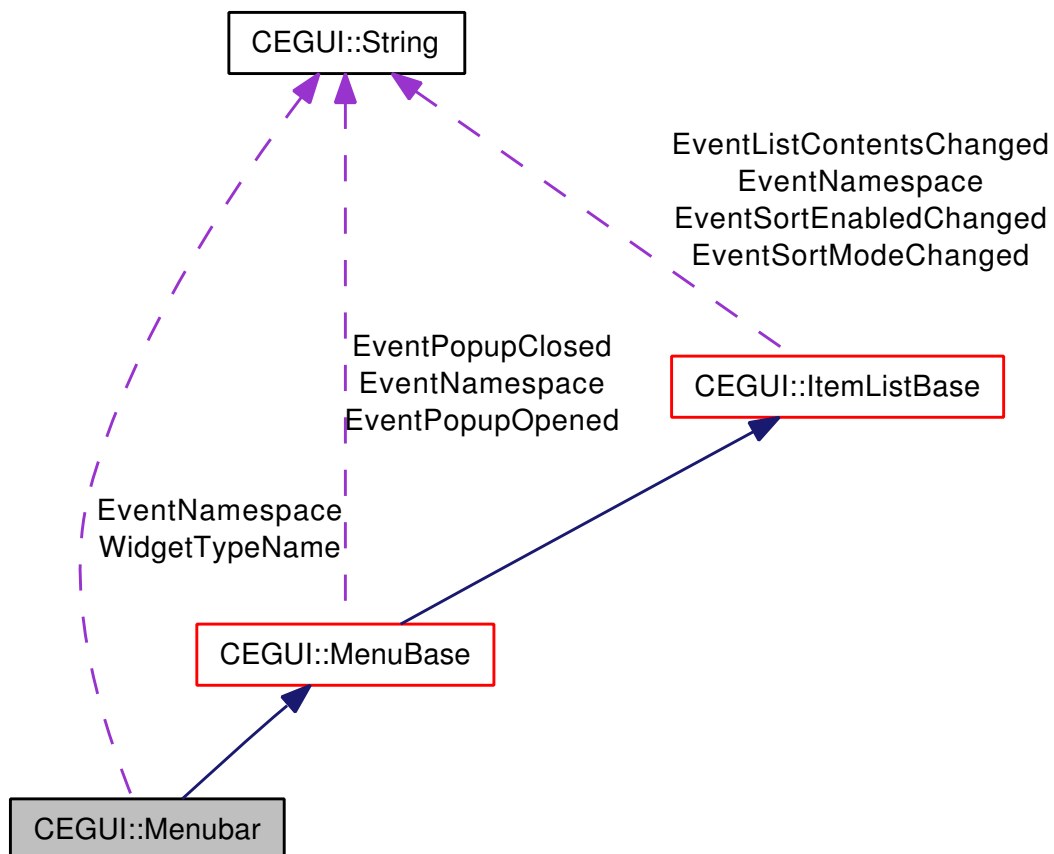
## 6.167 CEGUI::Menubar Class Reference

Base class for menu bars.

Inheritance diagram for CEGUI::Menubar:



Collaboration diagram for CEGUI::Menubar:



### Public Member Functions

- **Menubar** (const [String](#) &type, const [String](#) &name)  
*Constructor for [Menubar](#) objects.*

- virtual [~Menubar](#) (void)  
*Destructor for [Menubar](#) objects.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*

## Protected Member Functions

- virtual void [layoutItemWidgets](#) ()  
*Setup size and position for the item widgets attached to this [Menubar](#).*
- virtual [Size](#) [getContentSize](#) () const  
*Returns the [Size](#) in unclipped pixels of the content attached to this [ItemListBase](#) that is attached to it.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

### 6.167.1 Detailed Description

Base class for menu bars.

### 6.167.2 Member Function Documentation

#### 6.167.2.1 void CEGUI::Menubar::layoutItemWidgets (void) [protected, virtual]

Setup size and position for the item widgets attached to this [Menubar](#).

#### Returns:

Nothing.

Implements [CEGUI::ItemListBase](#).

#### 6.167.2.2 Size CEGUI::Menubar::getContentSize (void) const [protected, virtual]

Returns the [Size](#) in unclipped pixels of the content attached to this [ItemListBase](#) that is attached to it.

#### Returns:

[Size](#) object describing in unclipped pixels the size of the content ItemEntries attached to this menu.

Implements [CEGUI::ItemListBase](#).

**6.167.2.3** `virtual bool CEGUI::Menubar::testClassName_impl (const String & class_name) const`  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

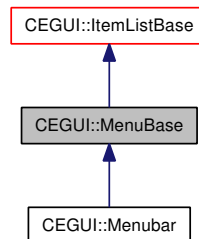
true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::MenuBase](#).

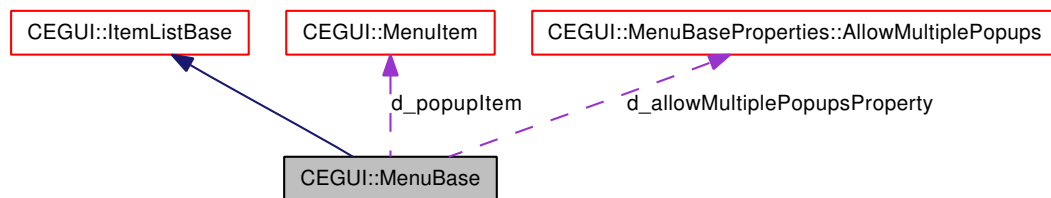
## 6.168 CEGUI::MenuBase Class Reference

Abstract base class for menus.

Inheritance diagram for CEGUI::MenuBase:



Collaboration diagram for CEGUI::MenuBase:



### Public Member Functions

- float [getItemSpacing](#) (void) const  
*Get the item spacing for this menu.*
- bool [isMultiplePopupsAllowed](#) (void) const  
*Return whether this menu allows multiple popup menus to open at the same time.*
- [MenuItem](#) \* [getPopupMenuItem](#) (void) const  
*Get currently opened MenuItem in this menu. Returns NULL if no menu item is open.*
- void [setItemSpacing](#) (float spacing)  
*Set the item spacing for this menu.*
- void [changePopupMenuItem](#) ([MenuItem](#) \*item)  
*Change the currently open MenuItem in this menu.*
- void [setAllowMultiplePopups](#) (bool setting)  
*Set whether this menu allows multiple popup menus to be opened simultaneously.*
- [MenuBase](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for MenuBase objects.*
- virtual [~MenuBase](#) (void)  
*Destructor for MenuBase objects.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [EventPopupOpened](#)  
*A [MenuItem](#) attached to this menu opened a [PopupMenu](#).*
- static const [String](#) [EventPopupClosed](#)  
*A [MenuItem](#) attached to this menu opened a [PopupMenu](#).*

## Protected Member Functions

- virtual void [onPopupOpened](#) ([WindowEventArgs](#) &e)  
*handler invoked internally when the a [MenuItem](#) attached to this menu opens its popup.*
- virtual void [onPopupClosed](#) ([WindowEventArgs](#) &e)  
*handler invoked internally when the a [MenuItem](#) attached to this menu closes its popup.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

## Protected Attributes

- float [d\\_itemSpacing](#)  
*The spacing in pixels between items.*
- [MenuItem](#) \* [d\\_popupItem](#)  
*The currently open [MenuItem](#). NULL if no item is open. If multiple popups are allowed, this means nothing.*
- bool [d\\_allowMultiplePopups](#)  
*true if multiple popup menus are allowed simultaneously. false if not.*

### 6.168.1 Detailed Description

Abstract base class for menus.

### 6.168.2 Member Function Documentation

#### 6.168.2.1 float CEGUI::MenuBase::getItemSpacing (void) const [inline]

Get the item spacing for this menu.

#### Returns:

A float value with the current item spacing for this menu

**6.168.2.2** `bool CEGUI::MenuBase::isMultiplePopupsAllowed (void) const` `[inline]`

Return whether this menu allows multiple popup menus to open at the same time.

**Returns:**

true if this menu allows multiple popup menus to be opened simultaneously. false if not

**6.168.2.3** `MenuItem* CEGUI::MenuBase::getPopupMenuItem (void) const` `[inline]`

Get currently opened [MenuItem](#) in this menu. Returns NULL if no menu item is open.

**Returns:**

Pointer to the [MenuItem](#) currently open.

**6.168.2.4** `void CEGUI::MenuBase::changePopupMenuItem (MenuItem * item)`

Change the currently open [MenuItem](#) in this menu.

**Parameters:**

*item* Pointer to a [MenuItem](#) to open or NULL to close any opened.

**6.168.2.5** `virtual bool CEGUI::MenuBase::testClassName_impl (const String & class_name) const`  
`[inline, protected, virtual]`

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

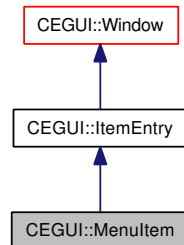
Reimplemented from [CEGUI::ItemListBase](#).

Reimplemented in [CEGUI::Menubar](#), and [CEGUI::PopupMenu](#).

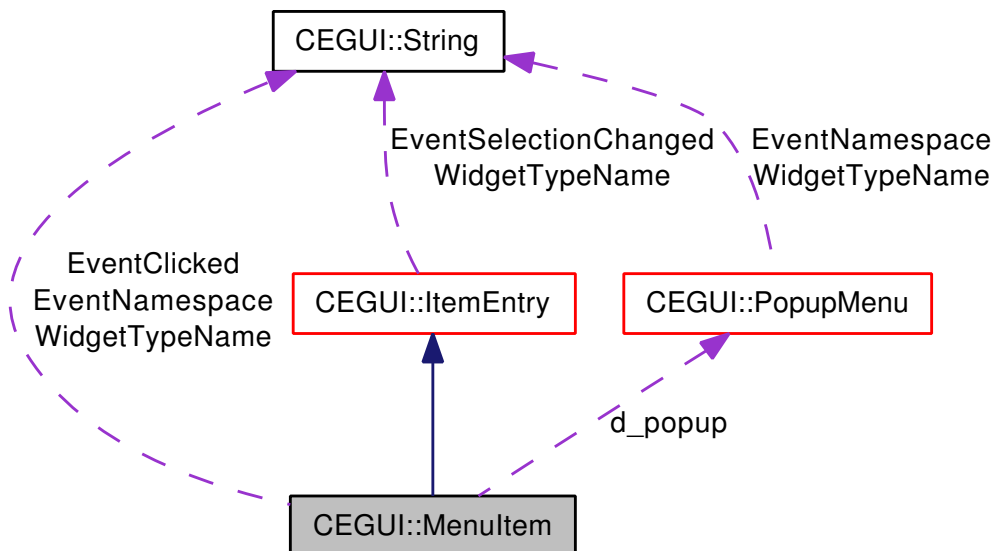
## 6.169 CEGUI::MenuItem Class Reference

Base class for menu items.

Inheritance diagram for CEGUI::MenuItem:



Collaboration diagram for CEGUI::MenuItem:



### Public Member Functions

- bool **isHovering** (void) const  
*return true if user is hovering over this widget (or it's pushed and user is not over it for highlight)*
- bool **isPushed** (void) const  
*Return true if the button widget is in the pushed state.*
- bool **isOpened** (void) const  
*Returns true if the popup menu attached to the menu item is open.*
- **PopupMenu** \* **getPopupMenu** (void) const  
*Get the **PopupMenu** that is currently attached to this **MenuItem**.*

- void [setPopupMenu](#) ([PopupMenu](#) \*popup)  
*Set the popup menu for this item.*
- void [openPopupMenu](#) (bool notify=true)  
*Opens the [PopupMenu](#).*
- void [closePopupMenu](#) (bool notify=true)  
*Closes the [PopupMenu](#).*
- bool [togglePopupMenu](#) (void)  
*Toggles the [PopupMenu](#).*
- [MenuItem](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [MenuItem](#) objects.*
- virtual [~MenuItem](#) (void)  
*Destructor for [MenuItem](#) objects.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventClicked](#)  
*The menu item was clicked.*

## Protected Member Functions

- virtual void [onClicked](#) ([WindowEventArgs](#) &e)  
*handler invoked internally when the [MenuItem](#) is clicked.*
- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onMouseDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void [onMouseLeaves](#) ([MouseEventArgs](#) &e)



*Handler called when the mouse cursor has left this window's area.*

- virtual void `onTextChanged` (`WindowEventArgs` &e)  
*Handler called when the window's text is changed.*
- void `updateInternalState` (const `Point` &mouse\_pos)  
*Update the internal state of the widget with the mouse at the given position.*
- void `closeAllMenuItemPopups` ()  
*Recursive function that closes all popups down the hierarchy starting with this one.*
- void `setPopupMenu_impl` (`PopupMenu` \*popup, bool add\_as\_child=true)  
*Set the popup menu for this item.*
- virtual bool `testClassName_impl` (const `String` &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

## Protected Attributes

- bool `d_pushed`  
*true when widget is pushed*
- bool `d_hovering`  
*true when the button is in 'hover' state and requires the hover rendering.*
- bool `d_opened`  
*true when the menu item's popup menu is in its opened state.*
- `PopupMenu` \* `d_popup`  
*PopupMenu that this item displays when activated.*
- bool `d_popupWasClosed`  
*Used internally to determine if a popup was just closed on a Clicked event.*

### 6.169.1 Detailed Description

Base class for menu items.

### 6.169.2 Member Function Documentation

#### 6.169.2.1 bool CEGUI::MenuItem::isHovering (void) const [inline]

return true if user is hovering over this widget (or it's pushed and user is not over it for highlight)

#### Returns:

true if the user is hovering or if the button is pushed and the mouse is not over the button. Otherwise return false.

**6.169.2.2** `bool CEGUI::MenuItem::isPushed (void) const` `[inline]`

Return true if the button widget is in the pushed state.

**Returns:**

true if the button-type widget is pushed, false if the widget is not pushed.

**6.169.2.3** `PopupMenu* CEGUI::MenuItem::getPopupMenu (void) const` `[inline]`

Get the [PopupMenu](#) that is currently attached to this [MenuItem](#).

**Returns:**

A pointer to the currently attached [PopupMenu](#). Null if there is no [PopupMenu](#) attached.

**6.169.2.4** `void CEGUI::MenuItem::setPopupMenu (PopupMenu * popup)`

Set the popup menu for this item.

**Parameters:**

*popup* popupmenu window to attach to this item

**Returns:**

Nothing.

**6.169.2.5** `void CEGUI::MenuItem::openPopupMenu (bool notify = true)`

Opens the [PopupMenu](#).

**Parameters:**

*notify* true if the parent menu bar or menu popup (if any) is to handle the open.

**6.169.2.6** `void CEGUI::MenuItem::closePopupMenu (bool notify = true)`

Closes the [PopupMenu](#).

**Parameters:**

*notify* true if the parent menubar (if any) is to handle the close.

**Returns:**

Nothing.

**6.169.2.7 bool CEGUI::MenuItem::togglePopupMenu (void)**

Toggles the [PopupMenu](#).

**Returns:**

true if the popup was opened. false if it was closed.

**6.169.2.8 void CEGUI::MenuItem::onMouseMove (MouseEventArgs & e) [protected, virtual]**

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.169.2.9 void CEGUI::MenuItem::onMouseButtonDown (MouseEventArgs & e) [protected, virtual]**

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.169.2.10 void CEGUI::MenuItem::onMouseButtonUp (MouseEventArgs & e) [protected, virtual]**

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.169.2.11 void CEGUI::MenuItem::onCaptureLost (WindowEventArgs & e) [protected, virtual]**

Handler called when this window loses capture of mouse inputs.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.169.2.12** `void CEGUI::MenuItem::onMouseLeaves (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has left this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.169.2.13** `void CEGUI::MenuItem::onTextChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.169.2.14** `void CEGUI::MenuItem::updateInternalState (const Point & mouse_pos)` [protected]

Update the internal state of the widget with the mouse at the given position.

**Parameters:**

*mouse\_pos* Point object describing, in screen pixel co-ordinates, the location of the mouse cursor.

**Returns:**

Nothing

**6.169.2.15** `void CEGUI::MenuItem::closeAllMenuItemPopups ()` [protected]

Recursive function that closes all popups down the hierarchy starting with this one.

**Returns:**

Nothing.

**6.169.2.16** `void CEGUI::MenuItem::setPopupMenu_impl (PopupMenu * popup, bool add_as_child = true)` [protected]

Set the popup menu for this item.

**Parameters:**

*popup* popupmenu window to attach to this item

**Returns:**

Nothing.

**6.169.2.17** `virtual bool CEGUI::MenuItem::testClassName_impl (const String & class_name)  
const [inline, protected, virtual]`

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

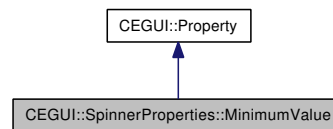
true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::ItemEntry](#).

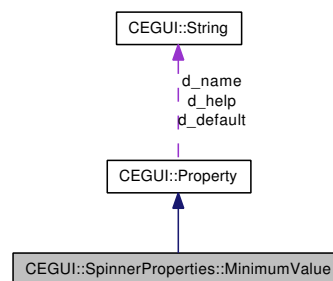
## 6.170 CEGUI::SpinnerProperties::MinimumValue Class Reference

[Property](#) to access the minimum value setting of the spinner.

Inheritance diagram for CEGUI::SpinnerProperties::MinimumValue:



Collaboration diagram for CEGUI::SpinnerProperties::MinimumValue:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.170.1 Detailed Description

[Property](#) to access the minimum value setting of the spinner.

#### Usage:

- Name: [MinimumValue](#)
- Format: "[float]".

#### Where:

- [float] represents the current minimum value of the [Spinner](#) widget.

## 6.170.2 Member Function Documentation

### 6.170.2.1 String CEGUI::SpinnerProperties::MinimumValue::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.170.2.2 void CEGUI::SpinnerProperties::MinimumValue::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

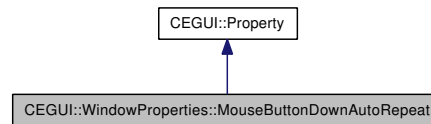
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

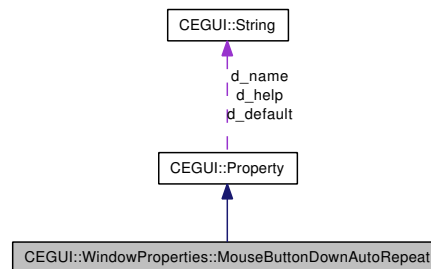
## 6.171 CEGUI::WindowProperties::MouseButtonDownAutoRepeat Class Reference

[Property](#) to control whether the window will receive autorepeat mouse button down events.

Inheritance diagram for CEGUI::WindowProperties::MouseButtonDownAutoRepeat:



Collaboration diagram for CEGUI::WindowProperties::MouseButtonDownAutoRepeat:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.171.1 Detailed Description

[Property](#) to control whether the window will receive autorepeat mouse button down events.

This property offers access to the setting that controls whether a window will receive autorepeat mouse button down events.

#### Usage:

- Name: [MouseButtonDownAutoRepeat](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the [Window](#) will receive autorepeat mouse button down events.
- "False" to indicate the [Window](#) will not receive autorepeat mouse button down events.



## 6.171.2 Member Function Documentation

### 6.171.2.1 String CEGUI::WindowProperties::MouseButtonDownAutoRepeat::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.171.2.2 void CEGUI::WindowProperties::MouseButtonDownAutoRepeat::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

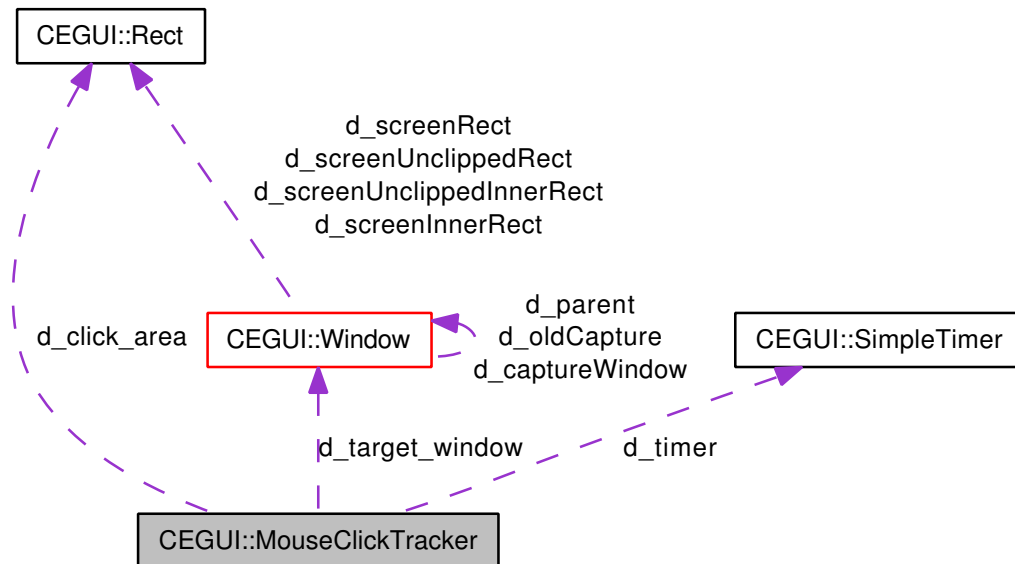
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.172 CEGUI::MouseClickedTracker Struct Reference

Implementation structure used in tracking up & down mouse button inputs in order to generate click, double-click, and triple-click events.

Collaboration diagram for CEGUI::MouseClickedTracker:



### Public Attributes

- [SimpleTimer d\\_timer](#)  
*Timer used to track clicks for this button.*
- [int d\\_click\\_count](#)  
*count of clicks made so far.*
- [Rect d\\_click\\_area](#)  
*area used to detect multi-clicks*
- [Window \\* d\\_target\\_window](#)  
*target window for any events generated.*

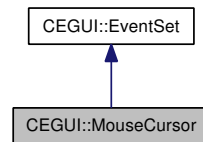
### 6.172.1 Detailed Description

Implementation structure used in tracking up & down mouse button inputs in order to generate click, double-click, and triple-click events.

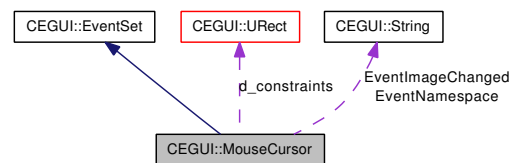
## 6.173 CEGUI::MouseButton Class Reference

Class that allows access to the GUI system mouse cursor.

Inheritance diagram for CEGUI::MouseButton:



Collaboration diagram for CEGUI::MouseButton:



### Public Member Functions

- **MouseButton** (void)  
*Constructor for **MouseButton** objects.*
- **~MouseButton** (void)  
*Destructor for **MouseButton** objects.*
- void **setImage** (const **String** &imageset, const **String** &image\_name)  
*Set the current mouse cursor image.*
- void **setImage** (const **Image** \*image)  
*Set the current mouse cursor image.*
- const **Image** \* **getImage** (void) const  
*Get the current mouse cursor image.*
- void **draw** (void) const  
*Makes the cursor draw itself.*
- void **setPosition** (const **Point** &position)  
*Set the current mouse cursor position.*
- void **offsetPosition** (const **Point** &offset)  
*Offset the mouse cursor position by the deltas specified in offset.*
- void **setConstraintArea** (const **Rect** \*area)  
*Set the area that the mouse cursor is constrained to.*

- void [setUnifiedConstraintArea](#) (const [URect](#) \*area)  
*Set the area that the mouse cursor is constrained to.*
- void [hide](#) (void)  
*Hides the mouse cursor.*
- void [show](#) (void)  
*Shows the mouse cursor.*
- void [setVisible](#) (bool visible)  
*Set the visibility of the mouse cursor.*
- bool [isVisible](#) (void) const  
*return whether the mouse cursor is visible.*
- [Point](#) [getPosition](#) (void) const  
*Return the current mouse cursor position as a pixel offset from the top-left corner of the display.*
- [Rect](#) [getConstraintArea](#) (void) const  
*return the current constraint area of the mouse cursor.*
- const [URect](#) & [getUnifiedConstraintArea](#) (void) const  
*return the current constraint area of the mouse cursor.*
- [Point](#) [getDisplayIndependantPosition](#) (void) const  
*Return the current mouse cursor position as display resolution independant values.*

## Static Public Member Functions

- static [MouseCursor](#) & [getSingleton](#) (void)  
*Return singleton [MouseCursor](#) object.*
- static [MouseCursor](#) \* [getSingletonPtr](#) (void)  
*Return pointer to singleton [MouseCursor](#) object.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [EventImageChanged](#)  
*The cursor image of the widget has changed.*

## Protected Member Functions

- virtual void [onImageChanged](#) ([MouseButtonEventArgs](#) &e)  
*event triggered internally when image of mouse cursor changes*

### 6.173.1 Detailed Description

Class that allows access to the GUI system mouse cursor.

The [MouseButton](#) provides functionality to access the position and imagery of the mouse cursor / pointer

### 6.173.2 Member Function Documentation

#### 6.173.2.1 MouseButton & CEGUI::MouseButton::getSingleton (void) [static]

Return singleton [MouseButton](#) object.

##### Returns:

Singleton [MouseButton](#) object

#### 6.173.2.2 MouseButton \* CEGUI::MouseButton::getSingletonPtr (void) [static]

Return pointer to singleton [MouseButton](#) object.

##### Returns:

Pointer to singleton [MouseButton](#) object

#### 6.173.2.3 void CEGUI::MouseButton::setImage (const String & *imageset*, const String & *image\_name*)

Set the current mouse cursor image.

##### Parameters:

*imageset* [String](#) object holding the name of the [Imageset](#) that contains the desired [Image](#).

*image\_name* [String](#) object holding the name of the desired [Image](#) on [Imageset](#) *imageset*.

##### Returns:

Nothing.

##### Exceptions:

[UnknownObjectException](#) thrown if *imageset* is not known, or if *imageset* contains no [Image](#) named *image\_name*.

**6.173.2.4** `const Image* CEGUI::MouseCursor::getImage (void) const` `[inline]`

Get the current mouse cursor image.

**Returns:**

The current image used to draw mouse cursor.

**6.173.2.5** `void CEGUI::MouseCursor::draw (void) const`

Makes the cursor draw itself.

**Returns:**

Nothing

**6.173.2.6** `void CEGUI::MouseCursor::setPosition (const Point & position)`

Set the current mouse cursor position.

**Parameters:**

*position* Point object describing the new location for the mouse. This will be clipped to within the renderer screen area.

**6.173.2.7** `void CEGUI::MouseCursor::offsetPosition (const Point & offset)`

Offset the mouse cursor position by the deltas specified in *offset*.

**Parameters:**

*offset* Point object which describes the amount to move the cursor in each axis.

**Returns:**

Nothing.

**6.173.2.8** `void CEGUI::MouseCursor::setConstraintArea (const Rect * area)`

Set the area that the mouse cursor is constrained to.

**Parameters:**

*area* Pointer to a [Rect](#) object that describes the area of the display that the mouse is allowed to occupy. The given area will be clipped to the current [Renderer](#) screen area - it is never possible for the mouse to leave this area. If this parameter is NULL, the constraint is set to the size of the current [Renderer](#) screen area.

**Returns:**

Nothing.

**6.173.2.9 void CEGUI::MouseButton::setUnifiedConstraintArea (const URect \* *area*)**

Set the area that the mouse cursor is constrained to.

**Parameters:**

*area* Pointer to a [URect](#) object that describes the area of the display that the mouse is allowed to occupy. The given area will be clipped to the current [Renderer](#) screen area - it is never possible for the mouse to leave this area. If this parameter is NULL, the constraint is set to the size of the current [Renderer](#) screen area.

**Returns:**

Nothing.

**6.173.2.10 void CEGUI::MouseButton::hide (void) [inline]**

Hides the mouse cursor.

**Returns:**

Nothing.

**6.173.2.11 void CEGUI::MouseButton::show (void) [inline]**

Shows the mouse cursor.

**Returns:**

Nothing.

**6.173.2.12 void CEGUI::MouseButton::setVisible (bool *visible*) [inline]**

Set the visibility of the mouse cursor.

**Parameters:**

*visible* 'true' to show the mouse cursor, 'false' to hide it.

**Returns:**

Nothing.

**6.173.2.13 bool CEGUI::MouseButton::isVisible (void) const [inline]**

return whether the mouse cursor is visible.

**Returns:**

true if the mouse cursor is visible, false if the mouse cursor is hidden.

**6.173.2.14 Point CEGUI::MouseCursor::getPosition (void) const** `[inline]`

Return the current mouse cursor position as a pixel offset from the top-left corner of the display.

**Returns:**

Point object describing the mouse cursor position in screen pixels.

**6.173.2.15 Rect CEGUI::MouseCursor::getConstraintArea (void) const**

return the current constraint area of the mouse cursor.

**Returns:**

[Rect](#) object describing the active area that the mouse cursor is constrained to.

**6.173.2.16 const URect & CEGUI::MouseCursor::getUnifiedConstraintArea (void) const**

return the current constraint area of the mouse cursor.

**Returns:**

[URect](#) object describing the active area that the mouse cursor is constrained to.

**6.173.2.17 Point CEGUI::MouseCursor::getDisplayIndependantPosition (void) const**

Return the current mouse cursor position as display resolution independant values.

**Returns:**

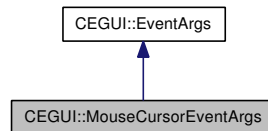
Point object describing the current mouse cursor position as resolution independant values that range from 0.0f to 1.0f, where 0.0f represents the left-most and top-most positions, and 1.0f represents the right-most and bottom-most positions.



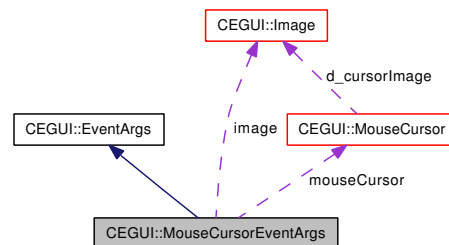
## 6.174 CEGUI::MouseCursorEventArgs Class Reference

[EventArgs](#) based class that is used for objects passed to input event handlers concerning mouse cursor events.

Inheritance diagram for CEGUI::MouseCursorEventArgs:



Collaboration diagram for CEGUI::MouseCursorEventArgs:



### Public Member Functions

- **MouseCursorEventArgs** ([MouseCursor](#) \*cursor)

### Public Attributes

- [MouseCursor](#) \* **mouseCursor**  
pointer to a [MouseCursor](#) object of relevance to the event.
- const [Image](#) \* **image**  
pointer to an [Image](#) object of relevance to the event.

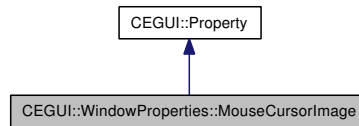
#### 6.174.1 Detailed Description

[EventArgs](#) based class that is used for objects passed to input event handlers concerning mouse cursor events.

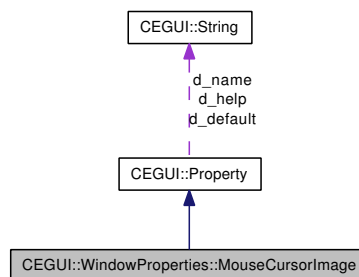
## 6.175 CEGUI::WindowProperties::MouseCursorImage Class Reference

[Property](#) to access window mouse cursor setting.

Inheritance diagram for CEGUI::WindowProperties::MouseCursorImage:



Collaboration diagram for CEGUI::WindowProperties::MouseCursorImage:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- bool **isDefault** (const [PropertyReceiver](#) \*receiver) const  
*Returns whether the property is at it's default value.*

### 6.175.1 Detailed Description

[Property](#) to access window mouse cursor setting.

This property offers access to the current mouse cursor image for the window.

#### Usage:

- Name: [MouseCursorImage](#)
- Format: "set:[text] image:[text]".

#### Where:

- set:[text] is the name of the [Imageset](#) containing the image. The [Imageset](#) name should not contain spaces. The [Imageset](#) specified must already be loaded.

- image:[text] is the name of the [Image](#) on the specified [Imageset](#). The [Image](#) name should not contain spaces.

## 6.175.2 Member Function Documentation

### 6.175.2.1 String CEGUI::WindowProperties::MouseCursorImage::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.175.2.2 void CEGUI::WindowProperties::MouseCursorImage::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

### 6.175.2.3 bool CEGUI::WindowProperties::MouseCursorImage::isDefault (const PropertyReceiver \* *receiver*) const [virtual]

Returns whether the property is at it's default value.

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

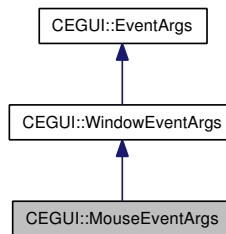
- true if the property has it's default value.
- false if the property has been modified from it's default value.

Reimplemented from [CEGUI::Property](#).

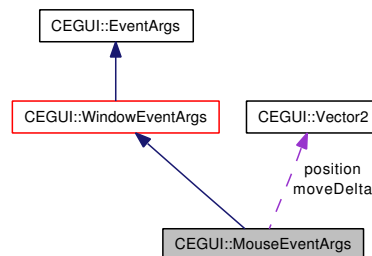
## 6.176 CEGUI::MouseEventArgs Class Reference

[EventArgs](#) based class that is used for objects passed to input event handlers concerning mouse input.

Inheritance diagram for CEGUI::MouseEventArgs:



Collaboration diagram for CEGUI::MouseEventArgs:



### Public Member Functions

- **MouseEventArgs** ([Window](#) \*wnd)

### Public Attributes

- [Point](#) **position**  
*holds current mouse position.*
- [Vector2](#) **moveDelta**  
*holds variation of mouse position from last mouse input*
- [MouseButton](#) **button**  
*one of the MouseButton enumerated values describing the mouse button causing the event (for button inputs only)*
- [uint](#) **sysKeys**  
*current state of the system keys and mouse buttons.*
- [float](#) **wheelChange**  
*Holds the amount the scroll wheel has changed.*
- [uint](#) **clickCount**

*Holds number of mouse button down events currently counted in a multi-click sequence (for button inputs only).*

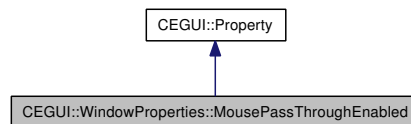
### 6.176.1 Detailed Description

[EventArgs](#) based class that is used for objects passed to input event handlers concerning mouse input.

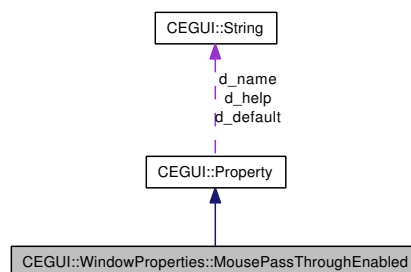
## 6.177 CEGUI::WindowProperties::MousePassThroughEnabled Class Reference

[Property](#) to access whether the window ignores mouse events and pass them through to any windows behind it.

Inheritance diagram for CEGUI::WindowProperties::MousePassThroughEnabled:



Collaboration diagram for CEGUI::WindowProperties::MousePassThroughEnabled:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.177.1 Detailed Description

[Property](#) to access whether the window ignores mouse events and pass them through to any windows behind it.

#### Usage:

- Name: [MousePassThroughEnabled](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the [Window](#) will not respond to mouse events but pass them directly to any children behind it.
- "False" to indicate the [Window](#) will respond to normally to all mouse events (Default).

## 6.177.2 Member Function Documentation

### 6.177.2.1 String CEGUI::WindowProperties::MousePassThroughEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.177.2.2 void CEGUI::WindowProperties::MousePassThroughEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

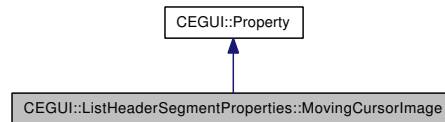
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

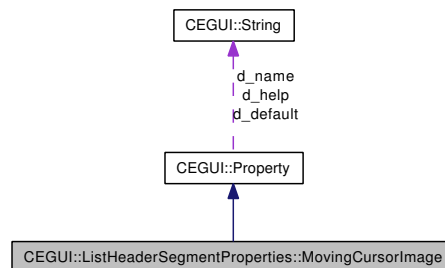
## 6.178 CEGUI::ListHeaderSegmentProperties::MovingCursorImage Class Reference

[Property](#) to access the segment moving cursor image.

Inheritance diagram for CEGUI::ListHeaderSegmentProperties::MovingCursorImage:



Collaboration diagram for CEGUI::ListHeaderSegmentProperties::MovingCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.178.1 Detailed Description

[Property](#) to access the segment moving cursor image.

Usage:

- Name: [MovingCursorImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.178.2 Member Function Documentation

#### 6.178.2.1 String CEGUI::ListHeaderSegmentProperties::MovingCursorImage::get (const [PropertyReceiver](#) \* receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).



**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.178.2.2** void CEGUI::ListHeaderSegmentProperties::MovingCursorImage::set  
(PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

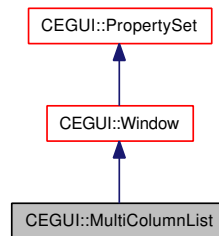
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

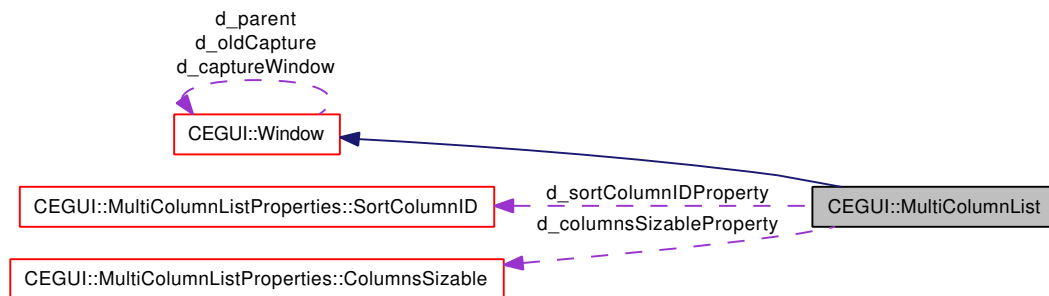
## 6.179 CEGUI::MultiColumnList Class Reference

Base class for the multi column list widget.

Inheritance diagram for CEGUI::MultiColumnList:



Collaboration diagram for CEGUI::MultiColumnList:



### Public Types

- enum [SelectionMode](#) {  
**RowSingle, RowMultiple, CellSingle, CellMultiple,**  
**NominatedColumnSingle, NominatedColumnMultiple, ColumnSingle, ColumnMultiple,**  
**NominatedRowSingle, NominatedRowMultiple }**

*Enumerated values for the selection modes possible with a Multi-column list.*

### Public Member Functions

- bool [isUserSortControlEnabled](#) (void) const  
*Return whether user manipulation of the sort column and direction are enabled.*
- bool [isUserColumnSizingEnabled](#) (void) const  
*Return whether the user may size column segments.*
- bool [isUserColumnDraggingEnabled](#) (void) const  
*Return whether the user may modify the order of the columns.*
- uint [getColumnCount](#) (void) const

*Return the number of columns in the multi-column list.*

- uint [getRowCount](#) (void) const  
*Return the number of rows in the multi-column list.*
- uint [getSortColumn](#) (void) const  
*Return the zero based index of the current sort column. There must be at least one column to successfully call this method.*
- uint [getColumnWithID](#) (uint col\_id) const  
*Return the zero based column index of the column with the specified ID.*
- uint [getColumnWithHeaderText](#) (const [String](#) &text) const  
*Return the zero based index of the column whos header text matches the specified text.*
- [UDim](#) [getTotalColumnHeadersWidth](#) (void) const  
*Return the total width of all column headers.*
- [UDim](#) [getColumnHeaderWidth](#) (uint col\_idx) const  
*Return the width of the specified column header (and therefore the column itself).*
- [ListHeaderSegment::SortDirection](#) [getSortDirection](#) (void) const  
*Return the currently set sort direction.*
- [ListHeaderSegment](#) & [getHeaderSegmentForColumn](#) (uint col\_idx) const  
*Return the [ListHeaderSegment](#) object for the specified column.*
- uint [getItemRowIndex](#) (const [ListboxItem](#) \*item) const  
*Return the zero based index of the Row that contains item.*
- uint [getItemColumnIndex](#) (const [ListboxItem](#) \*item) const  
*Return the current zero based index of the column that contains item.*
- [MCLGridRef](#) [getItemGridReference](#) (const [ListboxItem](#) \*item) const  
*Return the grid reference for item.*
- [ListboxItem](#) \* [getItemAtGridReference](#) (const [MCLGridRef](#) &grid\_ref) const  
*Return a pointer to the [ListboxItem](#) at the specified grid reference.*
- bool [isListboxItemInColumn](#) (const [ListboxItem](#) \*item, uint col\_idx) const  
*return whether [ListboxItem](#) item is attached to the column at index col\_idx.*
- bool [isListboxItemInRow](#) (const [ListboxItem](#) \*item, uint row\_idx) const  
*return whether [ListboxItem](#) item is attached to the row at index row\_idx.*
- bool [isListboxItemInList](#) (const [ListboxItem](#) \*item) const  
*return whether [ListboxItem](#) item is attached to the list box.*
- [ListboxItem](#) \* [findColumnItemWithText](#) (const [String](#) &text, uint col\_idx, const [ListboxItem](#) \*start\_item) const

Return the *ListboxItem* in column `col_idx` that has the text string `text`.

- *ListboxItem* \* `findRowItemWithText` (const *String* &`text`, uint `row_idx`, const *ListboxItem* \*`start_item`) const

Return the *ListboxItem* in row `row_idx` that has the text string `text`.

- *ListboxItem* \* `findListItemWithText` (const *String* &`text`, const *ListboxItem* \*`start_item`) const

Return the *ListboxItem* that has the text string `text`.

- *ListboxItem* \* `getFirstSelectedItem` (void) const

Return a pointer to the first selected *ListboxItem* attached to this list box.

- *ListboxItem* \* `getNextSelected` (const *ListboxItem* \*`start_item`) const

Return a pointer to the next selected *ListboxItem* after `start_item`.

- uint `getSelectedCount` (void) const

Return the number of selected *ListboxItems* attached to this list box.

- bool `isItemSelected` (const *MCLGridRef* &`grid_ref`) const

Return whether the *ListboxItem* at `grid_ref` is selected.

- uint `getNominatedSelectionColumnID` (void) const

Return the ID of the currently set nominated selection column to be used when in one of the *NominatedColumn\** selection modes.

- uint `getNominatedSelectionColumn` (void) const

Return the index of the currently set nominated selection column to be used when in one of the *NominatedColumn\** selection modes.

- uint `getNominatedSelectionRow` (void) const

Return the index of the currently set nominated selection row to be used when in one of the *NominatedRow\** selection modes.

- *MultiColumnList::SelectionMode* `getSelectionMode` (void) const

Return the currently set selection mode.

- bool `isVertScrollbarAlwaysShown` (void) const

Return whether the vertical scroll bar is always shown.

- bool `isHorzScrollbarAlwaysShown` (void) const

Return whether the horizontal scroll bar is always shown.

- uint `getColumnID` (uint `col_idx`) const

Return the ID code assigned to the requested column.

- uint `getRowID` (uint `row_idx`) const

Return the ID code assigned to the requested row.

- uint `getRowWithID` (uint `row_id`) const

Return the zero based row index of the row with the specified ID.

- [Rect](#) [getListRenderArea](#) (void) const  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- [Scrollbar](#) \* [getVertScrollbar](#) () const  
*Return a pointer to the vertical scrollbar component widget for this [MultiColumnList](#).*
- [Scrollbar](#) \* [getHorzScrollbar](#) () const  
*Return a pointer to the horizontal scrollbar component widget for this [MultiColumnList](#).*
- [ListHeader](#) \* [getListHeader](#) () const  
*Return a pointer to the list header component widget for this [MultiColumnList](#).*
- float [getTotalRowsHeight](#) (void) const  
*Return the sum of all row heights in pixels.*
- float [getWidestColumnItemWidth](#) (uint col\_idx) const  
*Return the pixel width of the widest item in the given column.*
- float [getHighestRowItemHeight](#) (uint row\_idx) const  
*Return, in pixels, the height of the highest item in the given row.*
- virtual void [initialiseComponents](#) (void)  
*Initialise the [Window](#) based object ready for use.*
- void [resetList](#) (void)  
*Remove all items from the list.*
- void [addColumn](#) (const [String](#) &text, uint col\_id, const [UDim](#) &width)  
*Add a column to the list box.*
- void [insertColumn](#) (const [String](#) &text, uint col\_id, const [UDim](#) &width, uint position)  
*Insert a new column in the list.*
- void [removeColumn](#) (uint col\_idx)  
*Removes a column from the list box. This will cause any [ListboxItem](#) using the autoDelete option in the column to be deleted.*
- void [removeColumnWithID](#) (uint col\_id)  
*Removes a column from the list box. This will cause any [ListboxItem](#) using the autoDelete option in the column to be deleted.*
- void [moveColumn](#) (uint col\_idx, uint position)  
*Move the column at index col\_idx so it is at index position.*
- void [moveColumnWithID](#) (uint col\_id, uint position)  
*Move the column with ID col\_id so it is at index position.*
- uint [addRow](#) (uint row\_id=0)  
*Add an empty row to the list box.*

- uint [addRow](#) ([ListboxItem](#) \*item, uint col\_id, uint row\_id=0)  
*Add a row to the list box, and set the item in the column with ID col\_id to item.*
- uint [insertRow](#) (uint row\_idx, uint row\_id=0)  
*Insert an empty row into the list box.*
- uint [insertRow](#) ([ListboxItem](#) \*item, uint col\_id, uint row\_idx, uint row\_id=0)  
*Insert a row into the list box, and set the item in the column with ID col\_id to item.*
- void [removeRow](#) (uint row\_idx)  
*Remove the list box row with index row\_idx. Any [ListboxItem](#) in row row\_idx using *autoDelete* mode will be deleted.*
- void [setItem](#) ([ListboxItem](#) \*item, const [MCLGridRef](#) &position)  
*Set the [ListboxItem](#) for grid reference position.*
- void [setItem](#) ([ListboxItem](#) \*item, uint col\_id, uint row\_idx)  
*Set the [ListboxItem](#) for the column with ID col\_id in row row\_idx.*
- void [setSelectionMode](#) ([MultiColumnList::SelectionMode](#) sel\_mode)  
*Set the selection mode for the list box.*
- void [setNominatedSelectionColumnID](#) (uint col\_id)  
*Set the column to be used for the *NominatedColumn\** selection modes.*
- void [setNominatedSelectionColumn](#) (uint col\_idx)  
*Set the column to be used for the *NominatedColumn\** selection modes.*
- void [setNominatedSelectionRow](#) (uint row\_idx)  
*Set the row to be used for the *NominatedRow\** selection modes.*
- void [setSortDirection](#) ([ListHeaderSegment::SortDirection](#) direction)  
*Set the sort direction to be used.*
- void [setSortColumn](#) (uint col\_idx)  
*Set the column to be used as the sort key.*
- void [setSortColumnByID](#) (uint col\_id)  
*Set the column to be used as the sort key.*
- void [setShowVertScrollbar](#) (bool setting)  
*Set whether the vertical scroll bar should always be shown, or just when needed.*
- void [setShowHorzScrollbar](#) (bool setting)  
*Set whether the horizontal scroll bar should always be shown, or just when needed.*
- void [clearAllSelections](#) (void)  
*Removed the selected state from any currently selected [ListboxItem](#) attached to the list.*

- void [setItemSelectState](#) ([ListboxItem](#) \*item, bool state)  
*Sets or clears the selected state of the given [ListboxItem](#) which must be attached to the list.*
- void [setItemSelectState](#) (const [MCLGridRef](#) &grid\_ref, bool state)  
*Sets or clears the selected state of the [ListboxItem](#) at the given grid reference.*
- void [handleUpdatedItemData](#) (void)  
*Inform the list box that one or more attached ListboxItems have been externally modified, and the list should re-sync its internal state and refresh the display as needed.*
- void [setColumnHeaderWidth](#) (uint col\_idx, const [UDim](#) &width)  
*Set the width of the specified column header (and therefore the column itself).*
- void [setUserSortControlEnabled](#) (bool setting)  
*Set whether user manipulation of the sort column and direction are enabled.*
- void [setUserColumnSizingEnabled](#) (bool setting)  
*Set whether the user may size column segments.*
- void [setUserColumnDraggingEnabled](#) (bool setting)  
*Set whether the user may modify the order of the columns.*
- void [autoSizeColumnHeader](#) (uint col\_idx)  
*Automatically determines the "best fit" size for the specified column and sets the column width to the same.*
- void [setRowID](#) (uint row\_idx, uint row\_id)  
*Set the ID code assigned to a given row.*
- [MultiColumnList](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for the Multi-column list base class.*
- virtual [~MultiColumnList](#) (void)  
*Destructor for the multi-column list base class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventSelectionModeChanged](#)  
*[Event](#) fired when the selection mode for the list box changes.*
- static const [String](#) [EventNominatedSelectColumnChanged](#)  
*[Event](#) fired when the nominated select column changes.*
- static const [String](#) [EventNominatedSelectRowChanged](#)

*Event fired when the nominated select row changes.*

- static const [String EventVertScrollbarModeChanged](#)  
*Event fired when the vertical scroll bar 'force' setting changes.*
- static const [String EventHorzScrollbarModeChanged](#)  
*Event fired when the horizontal scroll bar 'force' setting changes.*
- static const [String EventSelectionChanged](#)  
*Event fired when the current selection(s) within the list box changes.*
- static const [String EventListContentsChanged](#)  
*Event fired when the contents of the list box changes.*
- static const [String EventSortColumnChanged](#)  
*Event fired when the sort column changes.*
- static const [String EventSortDirectionChanged](#)  
*Event fired when the sort direction changes.*
- static const [String EventListColumnSized](#)  
*Event fired when the width of a column in the list changes.*
- static const [String EventListColumnMoved](#)  
*Event fired when the column order changes.*
- static const [String VertScrollbarNameSuffix](#)  
*Widget name suffix for the vertical scrollbar component.*
- static const [String HorzScrollbarNameSuffix](#)  
*Widget name suffix for the horizontal scrollbar component.*
- static const [String ListHeaderNameSuffix](#)  
*Widget name suffix for the list header component.*

## Protected Types

- typedef std::vector< [ListRow](#) > [ListItemGrid](#)

## Protected Member Functions

- void [configureScrollbars](#) (void)  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- bool [selectRange](#) (const [MCLGridRef](#) &start, const [MCLGridRef](#) &end)  
*select all strings between positions start and end. (inclusive). Returns true if something was modified.*



- bool [clearAllSelections\\_impl](#) (void)  
*Clear the selected state for all items (implementation).*
- [ListBoxItem](#) \* [getItemAtPoint](#) (const [Point](#) &pt) const  
*Return the [ListBoxItem](#) under the given window local pixel co-ordinate.*
- bool [setItemSelectState\\_impl](#) (const [MCLGridRef](#) grid\_ref, bool state)  
*Set select state for the given item. This appropriately selects other items depending upon the select mode. Returns true if something is changed, else false.*
- void [setSelectForItemsInRow](#) (uint row\_idx, bool state)  
*Set select state for all items in the given row.*
- void [setSelectForItemsInColumn](#) (uint col\_idx, bool state)  
*Set select state for all items in the given column.*
- void [moveColumn\\_impl](#) (uint col\_idx, uint position)  
*Move the column at index col\_idx so it is at index position. Implementation version which does not move the header segment (since that may have already happened).*
- bool [resetList\\_impl](#) (void)  
*Remove all items from the list.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- int [writePropertiesXML](#) ([XMLSerializer](#) &xml\_stream) const
- void [resortList](#) ()  
*Causes the internal list to be (re)sorted.*
- virtual void [onSelectionModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the selection mode of the list box changes.*
- virtual void [onNominatedSelectColumnChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the nominated selection column changes.*
- virtual void [onNominatedSelectRowChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the nominated selection row changes.*
- virtual void [onVertScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the vertical scroll bar 'force' mode is changed.*
- virtual void [onHorzScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the horizontal scroll bar 'force' mode is changed.*
- virtual void [onSelectionChanged](#) ([WindowEventArgs](#) &e)

*Handler called when the current selection changes.*

- virtual void **onListContentsChanged** ([WindowEventArgs](#) &e)  
*Handler called when the list contents is changed.*
- virtual void **onSortColumnChanged** ([WindowEventArgs](#) &e)  
*Handler called when the sort column changes.*
- virtual void **onSortDirectionChanged** ([WindowEventArgs](#) &e)  
*Handler called when the sort direction changes.*
- virtual void **onListColumnSized** ([WindowEventArgs](#) &e)  
*Handler called when a column is sized.*
- virtual void **onListColumnMoved** ([WindowEventArgs](#) &e)  
*Handler called when the column order is changed.*
- virtual void **onFontChanged** ([WindowEventArgs](#) &e)  
*Handler called when the window's font is changed.*
- virtual void **onSized** ([WindowEventArgs](#) &e)  
*Handler called when the window's size changes.*
- virtual void **onMouseButtonDown** ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void **onMouseWheel** ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*
- bool **handleHeaderScroll** (const [EventArgs](#) &e)
- bool **handleHeaderSegMove** (const [EventArgs](#) &e)
- bool **handleColumnSizeChange** (const [EventArgs](#) &e)
- bool **handleHorzScrollbar** (const [EventArgs](#) &e)
- bool **handleVertScrollbar** (const [EventArgs](#) &e)
- bool **handleSortColumnChange** (const [EventArgs](#) &e)
- bool **handleSortDirectionChange** (const [EventArgs](#) &e)
- bool **handleHeaderSegDbClick** (const [EventArgs](#) &e)

## Static Protected Member Functions

- static bool **pred\_descend** (const [ListRow](#) &a, const [ListRow](#) &b)  
*std algorithm predicate used for sorting in descending order*

## Protected Attributes

- bool **d\_forceVertScroll**  
*true if vertical scrollbar should always be displayed*

- bool [d\\_forceHorzScroll](#)  
*true if horizontal scrollbar should always be displayed*
- [SelectionMode](#) [d\\_selectMode](#)  
*Holds selection mode (represented by settings below).*
- uint [d\\_nominatedSelectCol](#)  
*Nominated column for single column selection.*
- uint [d\\_nominatedSelectRow](#)  
*Nominated row for single row selection.*
- bool [d\\_multiSelect](#)  
*Allow multiple selections.*
- bool [d\\_fullRowSelect](#)  
*All items in a row are selected.*
- bool [d\\_fullColSelect](#)  
*All items in a column are selected.*
- bool [d\\_useNominatedRow](#)  
*true if we use a nominated row to select.*
- bool [d\\_useNominatedCol](#)  
*true if we use a nominated col to select.*
- [ListBoxItem](#) \* [d\\_lastSelected](#)  
*holds pointer to the last selected item (used in range selections)*
- uint [d\\_columnCount](#)  
*keeps track of the number of columns.*
- [ListItemGrid](#) [d\\_grid](#)  
*Holds the list box data.*

## Friends

- class [MultiColumnListWindowRenderer](#)

## Classes

- struct [ListRow](#)  
*Struct used internally to represent a row in the list and also to ease sorting of the rows.*

### 6.179.1 Detailed Description

Base class for the multi column list widget.

## 6.179.2 Member Function Documentation

### 6.179.2.1 **bool CEGUI::MultiColumnList::isUserSortControlEnabled (void) const**

Return whether user manipulation of the sort column and direction are enabled.

**Returns:**

true if the user may interactively modify the sort column and direction. false if the user may not modify the sort column and direction (these can still be set programmatically).

### 6.179.2.2 **bool CEGUI::MultiColumnList::isUserColumnSizingEnabled (void) const**

Return whether the user may size column segments.

**Returns:**

true if the user may interactively modify the width of columns, false if they may not.

### 6.179.2.3 **bool CEGUI::MultiColumnList::isUserColumnDraggingEnabled (void) const**

Return whether the user may modify the order of the columns.

**Returns:**

true if the user may interactively modify the order of the columns, false if they may not.

### 6.179.2.4 **uint CEGUI::MultiColumnList::getColumnCount (void) const**

Return the number of columns in the multi-column list.

**Returns:**

uint value equal to the number of columns in the list.

### 6.179.2.5 **uint CEGUI::MultiColumnList::getRowCount (void) const**

Return the number of rows in the multi-column list.

**Returns:**

uint value equal to the number of rows currently in the list.

### 6.179.2.6 **uint CEGUI::MultiColumnList::getSortColumn (void) const**

Return the zero based index of the current sort column. There must be at least one column to successfully call this method.

**Returns:**

Zero based column index that is the current sort column.

**Exceptions:**

*InvalidRequestException* thrown if there are no columns in this multi column list.

**6.179.2.7 uint CEGUI::MultiColumnList::getColumnWithID (uint *col\_id*) const**

Return the zero based column index of the column with the specified ID.

**Parameters:**

*col\_id* ID code of the column whos index is to be returned.

**Returns:**

Zero based column index of the first column whos ID matches *col\_id*.

**Exceptions:**

*InvalidRequestException* thrown if no attached column has the requested ID.

**6.179.2.8 uint CEGUI::MultiColumnList::getColumnWithHeaderText (const String & *text*) const**

Return the zero based index of the column whos header text matches the specified text.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

**Returns:**

Zero based column index of the column whos header has the specified text.

**Exceptions:**

*InvalidRequestException* thrown if no columns header has the requested text.

**6.179.2.9 UDim CEGUI::MultiColumnList::getTotalColumnHeadersWidth (void) const**

Return the total width of all column headers.

**Returns:**

Sum total of all the column header widths as a [UDim](#).

**6.179.2.10 UDim CEGUI::MultiColumnList::getColumnHeaderWidth (uint *col\_idx*) const**

Return the width of the specified column header (and therefore the column itself).

**Parameters:**

*col\_idx* Zero based column index of the column whos width is to be returned.

**Returns:**

Width of the column header at the zero based column index specified by *col\_idx*, as a [UDim](#)

**Exceptions:**

[InvalidRequestException](#) thrown if *column* is out of range.

**6.179.2.11 ListHeaderSegment::SortDirection CEGUI::MultiColumnList::getSortDirection (void) const**

Return the currently set sort direction.

**Returns:**

One of the [ListHeaderSegment::SortDirection](#) enumerated values specifying the current sort direction.

**6.179.2.12 ListHeaderSegment & CEGUI::MultiColumnList::getHeaderSegmentForColumn (uint *col\_idx*) const**

Return the [ListHeaderSegment](#) object for the specified column.

**Parameters:**

*col\_idx* zero based index of the column whos [ListHeaderSegment](#) is to be returned.

**Returns:**

[ListHeaderSegment](#) object for the column at the requested index.

**Exceptions:**

[InvalidRequestException](#) thrown if *col\_idx* is out of range.

**6.179.2.13 uint CEGUI::MultiColumnList::getItemRowIndex (const ListboxItem \* *item*) const**

Return the zero based index of the Row that contains *item*.

**Parameters:**

*item* Pointer to the [ListboxItem](#) that the row index is to returned for.

**Returns:**

Zero based index of the row that contains [ListboxItem](#) *item*.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to the list box.

**6.179.2.14** `uint CEGUI::MultiColumnList::getItemColumnIndex (const ListboxItem * item) const`

Return the current zero based index of the column that contains *item*.

**Parameters:**

*item* Pointer to the [ListboxItem](#) that the column index is to returned for.

**Returns:**

Zero based index of the column that contains [ListboxItem](#) *item*.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to the list box.

**6.179.2.15** `MCLGridRef CEGUI::MultiColumnList::getItemGridReference (const ListboxItem * item) const`

Return the grid reference for *item*.

**Parameters:**

*item* Pointer to the [ListboxItem](#) whos current grid reference is to be returned.

**Returns:**

[MCLGridRef](#) object describing the current grid reference of [ListboxItem](#) *item*.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to the list box.

**6.179.2.16** `ListboxItem * CEGUI::MultiColumnList::getItemAtGridReference (const MCLGridRef & grid_ref) const`

Return a pointer to the [ListboxItem](#) at the specified grid reference.

**Parameters:**

*grid\_ref* [MCLGridRef](#) object that describes the position of the [ListboxItem](#) to be returned.

**Returns:**

Pointer to the [ListboxItem](#) at grid reference *grid\_ref*.

**Exceptions:**

[InvalidRequestException](#) thrown if *grid\_ref* is invalid for this list box.

**6.179.2.17** `bool CEGUI::MultiColumnList::isListBoxItemInColumn (const ListBoxItem * item,  
uint col_idx) const`

return whether [ListBoxItem](#) *item* is attached to the column at index *col\_idx*.

**Parameters:**

*item* Pointer to the [ListBoxItem](#) to look for.

*col\_idx* Zero based index of the column that is to be searched.

**Returns:**

- true if *item* is attached to list box column *col\_idx*.
- false if *item* is not attached to list box column *col\_idx*.

**Exceptions:**

[InvalidRequestException](#) thrown if *col\_idx* is out of range.

**6.179.2.18** `bool CEGUI::MultiColumnList::isListBoxItemInRow (const ListBoxItem * item, uint  
row_idx) const`

return whether [ListBoxItem](#) *item* is attached to the row at index *row\_idx*.

**Parameters:**

*item* Pointer to the [ListBoxItem](#) to look for.

*row\_idx* Zero based index of the row that is to be searched.

**Returns:**

- true if *item* is attached to list box row *row\_idx*.
- false if *item* is not attached to list box row *row\_idx*.

**Exceptions:**

[InvalidRequestException](#) thrown if *row\_idx* is out of range.

**6.179.2.19** `bool CEGUI::MultiColumnList::isListBoxItemInList (const ListBoxItem * item) const`

return whether [ListBoxItem](#) *item* is attached to the list box.

**Parameters:**

*item* Pointer to the [ListBoxItem](#) to look for.

**Returns:**

- true if *item* is attached to list box.
- false if *item* is not attached to list box.



**6.179.2.20** `ListboxItem * CEGUI::MultiColumnList::findColumnItemWithText (const String & text, uint col_idx, const ListboxItem * start_item) const`

Return the [ListboxItem](#) in column *col\_idx* that has the text string *text*.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

*col\_idx* Zero based index of the column to be searched.

*start\_item* Pointer to the [ListboxItem](#) where the exclusive search is to start, or NULL to search from the top of the column.

**Returns:**

Pointer to the first [ListboxItem](#) in column *col\_idx*, after *start\_item*, that has the string *text*.

**Exceptions:**

[InvalidRequestException](#) thrown if *start\_item* is not attached to the list box, or if *col\_idx* is out of range.

**6.179.2.21** `ListboxItem * CEGUI::MultiColumnList::findRowItemWithText (const String & text, uint row_idx, const ListboxItem * start_item) const`

Return the [ListboxItem](#) in row *row\_idx* that has the text string *text*.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

*row\_idx* Zero based index of the row to be searched.

*start\_item* Pointer to the [ListboxItem](#) where the exclusive search is to start, or NULL to search from the start of the row.

**Returns:**

Pointer to the first [ListboxItem](#) in row *row\_idx*, after *start\_item*, that has the string *text*.

**Exceptions:**

[InvalidRequestException](#) thrown if *start\_item* is not attached to the list box, or if *row\_idx* is out of range.

**6.179.2.22** `ListboxItem * CEGUI::MultiColumnList::findListItemWithText (const String & text, const ListboxItem * start_item) const`

Return the [ListboxItem](#) that has the text string *text*.

**Note:**

List box searching progresses across the columns in each row.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

*start\_item* Pointer to the [ListBoxItem](#) where the exclusive search is to start, or NULL to search the whole list box.

**Returns:**

Pointer to the first [ListBoxItem](#), after *start\_item*, that has the string *text*.

**Exceptions:**

[InvalidRequestException](#) thrown if *start\_item* is not attached to the list box.

**6.179.2.23 [ListBoxItem](#) \* CEGUI::MultiColumnList::getFirstSelectedItem (void) const**

Return a pointer to the first selected [ListBoxItem](#) attached to this list box.

**Note:**

List box searching progresses across the columns in each row.

**Returns:**

Pointer to the first [ListBoxItem](#) attached to this list box that is selected, or NULL if no item is selected.

**6.179.2.24 [ListBoxItem](#) \* CEGUI::MultiColumnList::getNextSelected (const [ListBoxItem](#) \* *start\_item*) const**

Return a pointer to the next selected [ListBoxItem](#) after *start\_item*.

**Note:**

List box searching progresses across the columns in each row.

**Parameters:**

*start\_item* Pointer to the [ListBoxItem](#) where the exclusive search is to start, or NULL to search the whole list box.

**Returns:**

Pointer to the first selected [ListBoxItem](#) attached to this list box, after *start\_item*, or NULL if no item is selected.

**Exceptions:**

[InvalidRequestException](#) thrown if *start\_item* is not attached to the list box.

**6.179.2.25 uint CEGUI::MultiColumnList::getSelectedCount (void) const**

Return the number of selected [ListBoxItems](#) attached to this list box.

return uint value equal to the number of [ListBoxItems](#) attached to this list box that are currently selected.

**6.179.2.26 bool CEGUI::MultiColumnList::isSelected (const MCLGridRef & grid\_ref) const**

Return whether the [ListBoxItem](#) at *grid\_ref* is selected.

**Parameters:**

*grid\_ref* [MCLGridRef](#) object describing the grid reference that is to be examined.

**Returns:**

- true if there is a [ListBoxItem](#) at *grid\_ref* and it is selected.
- false if there is no [ListBoxItem](#) at *grid\_ref*, or if the item is not selected.

**Exceptions:**

[InvalidRequestException](#) thrown if *grid\_ref* contains an invalid grid position.

**6.179.2.27 uint CEGUI::MultiColumnList::getNominatedSelectionColumnID (void) const**

Return the ID of the currently set nominated selection column to be used when in one of the NominatedColumn\* selection modes.

**Note:**

You should only ever call this when [getColumnCount\(\)](#) returns > 0.

**Returns:**

ID code of the nominated selection column.

**6.179.2.28 uint CEGUI::MultiColumnList::getNominatedSelectionColumn (void) const**

Return the index of the currently set nominated selection column to be used when in one of the NominatedColumn\* selection modes.

**Returns:**

Zero based index of the nominated selection column.

**6.179.2.29 uint CEGUI::MultiColumnList::getNominatedSelectionRow (void) const**

Return the index of the currently set nominated selection row to be used when in one of the NominatedRow\* selection modes.

**Returns:**

Zero based index of the nominated selection column.

**6.179.2.30   MultiColumnList::SelectionMode CEGUI::MultiColumnList::getSelectionMode (void) const**

Return the currently set selection mode.

**Returns:**

One of the [MultiColumnList::SelectionMode](#) enumerated values specifying the current selection mode.

**6.179.2.31   bool CEGUI::MultiColumnList::isVertScrollbarAlwaysShown (void) const**

Return whether the vertical scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.179.2.32   bool CEGUI::MultiColumnList::isHorzScrollbarAlwaysShown (void) const**

Return whether the horizontal scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.179.2.33   uint CEGUI::MultiColumnList::getColumnID (uint *col\_idx*) const**

Return the ID code assigned to the requested column.

**Parameters:**

*col\_idx* Zero based index of the column whos ID code is to be returned.

**Returns:**

Current ID code assigned to the column at the requested index.

**Exceptions:**

[InvalidRequestException](#) thrown if *col\_idx* is out of range

**6.179.2.34   uint CEGUI::MultiColumnList::getRowID (uint *row\_idx*) const**

Return the ID code assigned to the requested row.

**Parameters:**

*row\_idx* Zero based index of the row who's ID code is to be returned.

**Returns:**

Current ID code assigned to the row at the requested index.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if *row\_idx* is out of range

**6.179.2.35** `uint CEGUI::MultiColumnList::getRowWithID (uint row_id) const`

Return the zero based row index of the row with the specified ID.

**Parameters:**

*row\_id* ID code of the row who's index is to be returned.

**Returns:**

Zero based row index of the first row who's ID matches *row\_id*.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if no row has the requested ID.

**6.179.2.36** `Rect CEGUI::MultiColumnList::getListRenderArea (void) const`

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

**6.179.2.37** `Scrollbar * CEGUI::MultiColumnList::getVertScrollbar () const`

Return a pointer to the vertical scrollbar component widget for this [MultiColumnList](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the vertical [Scrollbar](#) component does not exist.

**6.179.2.38** `Scrollbar * CEGUI::MultiColumnList::getHorzScrollbar () const`

Return a pointer to the horizontal scrollbar component widget for this [MultiColumnList](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the horizontal [Scrollbar](#) component does not exist.

**6.179.2.39** `ListHeader * CEGUI::MultiColumnList::getListHeader () const`

Return a pointer to the list header component widget for this [MultiColumnList](#).

**Returns:**

Pointer to a [ListHeader](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the list header component does not exist.

**6.179.2.40** `void CEGUI::MultiColumnList::initialiseComponents (void) [virtual]`

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.179.2.41** `void CEGUI::MultiColumnList::resetList (void)`

Remove all items from the list.

Note that this will cause 'AutoDelete' items to be deleted.

**6.179.2.42** `void CEGUI::MultiColumnList::addColumn (const String & text, uint col_id, const UDim & width)`

Add a column to the list box.

**Parameters:**

*text* [String](#) object containing the text label for the column header.

*col\_id* ID code to be assigned to the column header.

*width* [UDim](#) describing the initial width to be set for the column.

**Returns:**

Nothing.

**6.179.2.43 void CEGUI::MultiColumnList::insertColumn (const String & *text*, uint *col\_id*, const UDim & *width*, uint *position*)**

Insert a new column in the list.

**Parameters:**

*text* [String](#) object containing the text label for the column header.

*col\_id* ID code to be assigned to the column header.

*width* [UDim](#) describing the initial width to be set for the column.

*position* Zero based index where the column is to be inserted. If this is greater than the current number of columns, the new column is inserted at the end.

**Returns:**

Nothing.

**6.179.2.44 void CEGUI::MultiColumnList::removeColumn (uint *col\_idx*)**

Removes a column from the list box. This will cause any [ListboxItem](#) using the autoDelete option in the column to be deleted.

**Parameters:**

*col\_idx* Zero based index of the column to be removed.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *col\_idx* is invalid.

**6.179.2.45 void CEGUI::MultiColumnList::removeColumnWithID (uint *col\_id*)**

Removes a column from the list box. This will cause any [ListboxItem](#) using the autoDelete option in the column to be deleted.

**Parameters:**

*col\_id* ID code of the column to be deleted.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if no column with *col\_id* is available on this list box.

**6.179.2.46 void CEGUI::MultiColumnList::moveColumn (uint *col\_idx*, uint *position*)**

Move the column at index *col\_idx* so it is at index *position*.

**Parameters:**

*col\_idx* Zero based index of the column to be moved.

*position* Zero based index of the new position for the column.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *col\_idx* is invalid.

**6.179.2.47 void CEGUI::MultiColumnList::moveColumnWithID (uint *col\_id*, uint *position*)**

Move the column with ID *col\_id* so it is at index *position*.

**Parameters:**

*col\_id* ID code of the column to be moved.

*position* Zero based index of the new position for the column.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if no column with *col\_id* is available on this list box.

**6.179.2.48 uint CEGUI::MultiColumnList::addRow (uint *row\_id* = 0)**

Add an empty row to the list box.

**Parameters:**

*row\_id* ID code to be assigned to the new row.

**Note:**

If the list is being sorted, the new row will appear at an appropriate position according to the sorting being applied. If no sorting is being done, the new row will appear at the bottom of the list.

**Returns:**

Initial zero based index of the new row.



**6.179.2.49** `uint CEGUI::MultiColumnList::addRow (ListBoxItem * item, uint col_id, uint row_id = 0)`

Add a row to the list box, and set the item in the column with ID *col\_id* to *item*.

**Note:**

If the list is being sorted, the new row will appear at an appropriate position according to the sorting being applied. If no sorting is being done, the new row will appear at the bottom of the list.

**Parameters:**

*item* Pointer to a [ListBoxItem](#) to be used as the initial contents for the column with ID *col\_id*.

*col\_id* ID code of the column whos initial item is to be set to *item*.

*row\_id* ID code to be assigned to the new row.

**Returns:**

Initial zero based index of the new row.

**Exceptions:**

[InvalidRequestException](#) thrown if no column with the specified ID is attached to the list box.

**6.179.2.50** `uint CEGUI::MultiColumnList::insertRow (uint row_idx, uint row_id = 0)`

Insert an empty row into the list box.

**Note:**

If the list is being sorted, the new row will appear at an appropriate position according to the sorting being applied. If no sorting is being done, the new row will appear at the specified index.

**Parameters:**

*row\_idx* Zero based index where the row should be inserted. If this is greater than the current number of rows, the row is appended to the list.

*row\_id* ID code to be assigned to the new row.

**Returns:**

Zero based index where the row was actually inserted.

**6.179.2.51** `uint CEGUI::MultiColumnList::insertRow (ListBoxItem * item, uint col_id, uint row_idx, uint row_id = 0)`

Insert a row into the list box, and set the item in the column with ID *col\_id* to *item*.

**Note:**

If the list is being sorted, the new row will appear at an appropriate position according to the sorting being applied. If no sorting is being done, the new row will appear at the specified index.

**Parameters:**

- item* Pointer to a [ListBoxItem](#) to be used as the initial contents for the column with ID *col\_id*.
- col\_id* ID code of the column whos initial item is to be set to *item*.
- row\_idx* Zero based index where the row should be inserted. If this is greater than the current number of rows, the row is appended to the list.
- row\_id* ID code to be assigned to the new row.

**Returns:**

Zero based index where the row was actually inserted.

**Exceptions:**

[InvalidRequestException](#) thrown if no column with the specified ID is attached to the list box.

**6.179.2.52 void CEGUI::MultiColumnList::removeRow (uint row\_idx)**

Remove the list box row with index *row\_idx*. Any [ListBoxItem](#) in row *row\_idx* using autoDelete mode will be deleted.

**Parameters:**

*row\_idx* Zero based index of the row to be removed.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *row\_idx* is invalid.

**6.179.2.53 void CEGUI::MultiColumnList::setItem (ListBoxItem \* item, const MCLGridRef & position)**

Set the [ListBoxItem](#) for grid reference *position*.

**Parameters:**

- item* Pointer to the [ListBoxItem](#) to be set at *position*.
- position* [MCLGridRef](#) describing the grid reference of the item to be set.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *position* contains an invalid grid reference.

**6.179.2.54 void CEGUI::MultiColumnList::setItem (ListBoxItem \* *item*, uint *col\_id*, uint *row\_idx*)**

Set the [ListBoxItem](#) for the column with ID *col\_id* in row *row\_idx*.

**Parameters:**

*item* Pointer to the [ListBoxItem](#) to be set into the list.

*col\_id* ID code of the column to receive *item*.

*row\_idx* Zero based index of the row to receive *item*.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if no column with ID *col\_id* exists, or of *row\_idx* is out of range.

**6.179.2.55 void CEGUI::MultiColumnList::setSelectionMode (MultiColumnList::SelectionMode *sel\_mode*)**

Set the selection mode for the list box.

**Parameters:**

*sel\_mode* One of the [MultiColumnList::SelectionMode](#) enumerated values specifying the selection mode to be used.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if the value specified for *sel\_mode* is invalid.

**6.179.2.56 void CEGUI::MultiColumnList::setNominatedSelectionColumnID (uint *col\_id*)**

Set the column to be used for the NominatedColumn\* selection modes.

**Parameters:**

*col\_id* ID code of the column to be used in NominatedColumn\* selection modes.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if no column has ID code *col\_id*.

**6.179.2.57 void CEGUI::MultiColumnList::setNominatedSelectionColumn (uint *col\_idx*)**

Set the column to be used for the NominatedColumn\* selection modes.

**Parameters:**

*col\_idx* zero based index of the column to be used in NominatedColumn\* selection modes.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *col\_idx* is out of range.

**6.179.2.58 void CEGUI::MultiColumnList::setNominatedSelectionRow (uint *row\_idx*)**

Set the row to be used for the NominatedRow\* selection modes.

**Parameters:**

*row\_idx* zero based index of the row to be used in NominatedRow\* selection modes.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *row\_idx* is out of range.

**6.179.2.59 void CEGUI::MultiColumnList::setSortDirection (ListHeaderSegment::SortDirection *direction*)**

Set the sort direction to be used.

**Parameters:**

*direction* One of the [ListHeaderSegment::SortDirection](#) enumerated values specifying the sort direction to be used.

**Returns:**

Nothing.

**6.179.2.60 void CEGUI::MultiColumnList::setSortColumn (uint *col\_idx*)**

Set the column to be used as the sort key.

**Parameters:**

*col\_idx* Zero based index of the column to use as the key when sorting the list items.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if col\_idx is out of range.

**6.179.2.61 void CEGUI::MultiColumnList::setSortColumnByID (uint col\_id)**

Set the column to be used as the sort key.

**Parameters:**

*col\_id* ID code of the column to use as the key when sorting the list items.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if col\_id is invalid for this list box.

**6.179.2.62 void CEGUI::MultiColumnList::setShowVertScrollbar (bool setting)**

Set whether the vertical scroll bar should always be shown, or just when needed.

**Parameters:**

- setting*
- true to have the vertical scroll bar shown at all times.
  - false to have the vertical scroll bar appear only when needed.

**Returns:**

Nothing.

**6.179.2.63 void CEGUI::MultiColumnList::setShowHorzScrollbar (bool setting)**

Set whether the horizontal scroll bar should always be shown, or just when needed.

**Parameters:**

- setting*
- true to have the horizontal scroll bar shown at all times.
  - false to have the horizontal scroll bar appear only when needed.

**Returns:**

Nothing.

**6.179.2.64 void CEGUI::MultiColumnList::clearAllSelections (void)**

Removed the selected state from any currently selected [ListboxItem](#) attached to the list.

**Returns:**

Nothing.

**6.179.2.65 void CEGUI::MultiColumnList::setItemSelectState (ListboxItem \* item, bool state)**

Sets or clears the selected state of the given [ListboxItem](#) which must be attached to the list.

**Note:**

Depending upon the current selection mode, this may cause other items to be selected, other items to be deselected, or for nothing to actually happen at all.

**Parameters:**

- item* Pointer to the attached [ListboxItem](#) to be affected.
- state*
- true to put the [ListboxItem](#) into the selected state.
  - false to put the [ListboxItem](#) into the de-selected state.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to the list box.

**6.179.2.66 void CEGUI::MultiColumnList::setItemSelectState (const MCLGridRef & grid\_ref, bool state)**

Sets or clears the selected state of the [ListboxItem](#) at the given grid reference.

**Note:**

Depending upon the current selection mode, this may cause other items to be selected, other items to be deselected, or for nothing to actually happen at all.

**Parameters:**

- grid\_ref* [MCLGridRef](#) object describing the position of the item to be affected.
- state*
- true to put the [ListboxItem](#) into the selected state.
  - false to put the [ListboxItem](#) into the de-selected state.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *grid\_ref* is invalid for this list box.

**6.179.2.67 void CEGUI::MultiColumnList::handleUpdatedItemData (void)**

Inform the list box that one or more attached ListboxItems have been externally modified, and the list should re-sync its internal state and refresh the display as needed.

**Returns:**

Nothing.

**6.179.2.68 void CEGUI::MultiColumnList::setColumnHeaderWidth (uint *col\_idx*, const UDim & *width*)**

Set the width of the specified column header (and therefore the column itself).

**Parameters:**

*col\_idx* Zero based column index of the column whos width is to be set.  
*width* UDim value specifying the new width for the column.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *column* is out of range.

**6.179.2.69 void CEGUI::MultiColumnList::setUserSortControlEnabled (bool *setting*)**

Set whether user manipulation of the sort column and direction are enabled.

**Parameters:**

*setting*

- true if the user may interactively modify the sort column and direction.
- false if the user may not modify the sort column and direction (these can still be set programmatically).

**Returns:**

Nothing.

**6.179.2.70 void CEGUI::MultiColumnList::setUserColumnSizingEnabled (bool *setting*)**

Set whether the user may size column segments.

**Parameters:**

*setting*

- true if the user may interactively modify the width of columns.
- false if the user may not change the width of the columns.

**Returns:**

Nothing.

**6.179.2.71 void CEGUI::MultiColumnList::setUserColumnDraggingEnabled (bool *setting*)**

Set whether the user may modify the order of the columns.

**Parameters:**

- setting* • true if the user may interactively modify the order of the columns.
- false if the user may not modify the order of the columns.

**6.179.2.72 void CEGUI::MultiColumnList::autoSizeColumnHeader (uint *col\_idx*)**

Automatically determines the "best fit" size for the specified column and sets the column width to the same.

**Parameters:**

*col\_idx* Zero based index of the column to be sized.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *col\_idx* is out of range.

**6.179.2.73 void CEGUI::MultiColumnList::setRowID (uint *row\_idx*, uint *row\_id*)**

Set the ID code assigned to a given row.

**Parameters:**

*row\_idx* Zero based index of the row who's ID code is to be set.

*row\_id* ID code to be assigned to the row at the requested index.

**Returns:**

Nothing.

**Exceptions:**

*InvalidRequestException* thrown if *row\_idx* is out of range

**6.179.2.74 void CEGUI::MultiColumnList::configureScrollbars (void) [protected]**

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

display required integrated scroll bars according to current state of the list box and update their values.



**6.179.2.75** `bool CEGUI::MultiColumnList::clearAllSelections_impl (void)` [protected]

Clear the selected state for all items (implementation).

**Returns:**

true if some selections were cleared, false nothing was changed.

**6.179.2.76** `ListboxItem * CEGUI::MultiColumnList::getItemAtPoint (const Point & pt) const`  
[protected]

Return the [ListboxItem](#) under the given window local pixel co-ordinate.

**Returns:**

[ListboxItem](#) that is under window pixel co-ordinate *pt*, or NULL if no item is under that position.

**6.179.2.77** `void CEGUI::MultiColumnList::moveColumn_impl (uint col_idx, uint position)`  
[protected]

Move the column at index *col\_idx* so it is at index *position*. Implementation version which does not move the header segment (since that may have already happened).

**Exceptions:**

[InvalidRequestException](#) thrown if *col\_idx* is invalid.

**6.179.2.78** `bool CEGUI::MultiColumnList::resetList_impl (void)` [protected]

Remove all items from the list.

**Note:**

Note that this will cause 'AutoDelete' items to be deleted.

**Returns:**

- true if the list contents were changed.
- false if the list contents were not changed (list already empty).

**6.179.2.79** `virtual bool CEGUI::MultiColumnList::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.179.2.80** `virtual bool CEGUI::MultiColumnList::validateWindowRenderer (const String & name) const` [inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.179.2.81** `void CEGUI::MultiColumnList::onFontChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's font is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.179.2.82** `void CEGUI::MultiColumnList::onSized (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's size changes.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.179.2.83** `void CEGUI::MultiColumnList::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

- e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.179.2.84** `void CEGUI::MultiColumnList::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

## 6.180 CEGUI::MultiColumnList::ListRow Struct Reference

Struct used internally to represent a row in the list and also to ease sorting of the rows.

### Public Types

- typedef std::vector< [ListBoxItem](#) \* > **RowItems**

### Public Member Functions

- [ListBoxItem](#) \*const & **operator**[ ] (uint idx) const
- [ListBoxItem](#) \*& **operator**[ ] (uint idx)
- bool **operator**< (const [ListRow](#) &rhs) const
- bool **operator**> (const [ListRow](#) &rhs) const

### Public Attributes

- RowItems **d\_items**
- uint **d\_sortColumn**
- uint **d\_rowID**

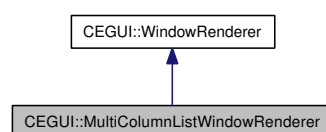
#### 6.180.1 Detailed Description

Struct used internally to represent a row in the list and also to ease sorting of the rows.

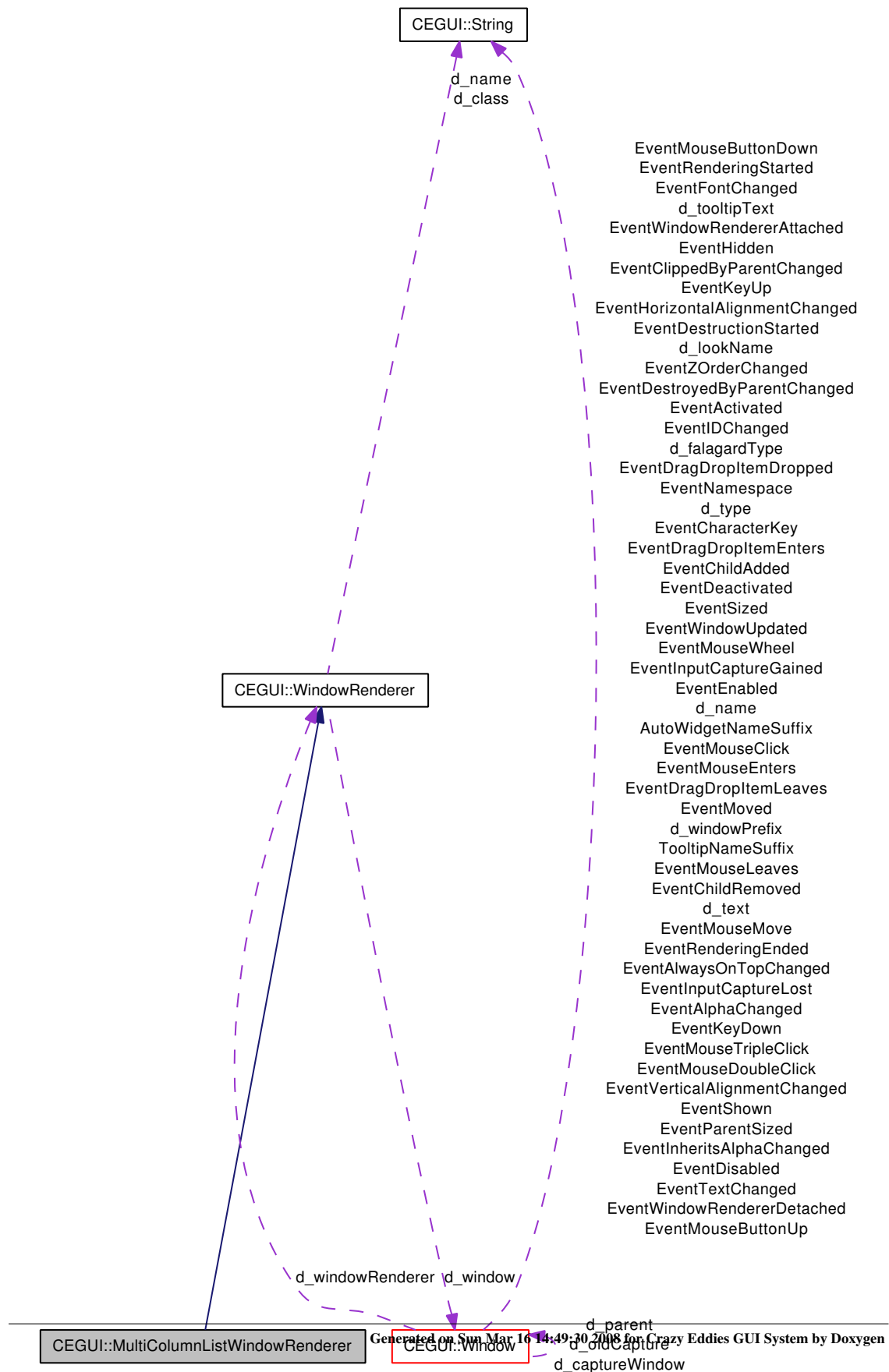
## 6.181 CEGUI::MultiColumnListWindowRenderer Class Reference

Base class for the multi column list window renderer.

Inheritance diagram for CEGUI::MultiColumnListWindowRenderer:



Collaboration diagram for CEGUI::MultiColumnListWindowRenderer:



## Public Member Functions

- [MultiColumnListWindowRenderer](#) (const [String](#) &name)  
*Constructor.*
- virtual [Rect](#) [getListRenderArea](#) (void) const =0  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*

### 6.181.1 Detailed Description

Base class for the multi column list window renderer.

### 6.181.2 Member Function Documentation

**6.181.2.1** virtual [Rect](#) CEGUI::MultiColumnListWindowRenderer::getListRenderArea (void)  
**const** [pure virtual]

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

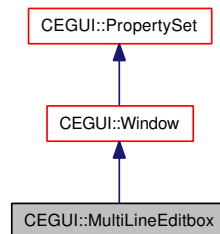
#### Returns:

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

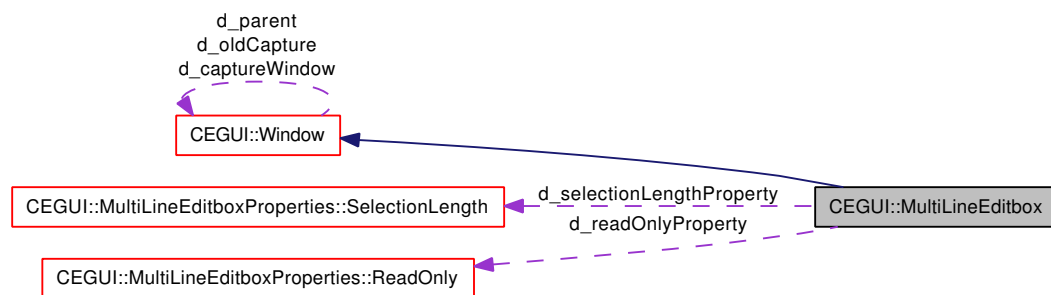
## 6.182 CEGUI::MultiLineEditbox Class Reference

Base class for the multi-line edit box widget.

Inheritance diagram for CEGUI::MultiLineEditbox:



Collaboration diagram for CEGUI::MultiLineEditbox:



### Public Types

- typedef std::vector< [LineInfo](#) > [LineList](#)  
Type for collection of LineInfos.

### Public Member Functions

- bool [hasInputFocus](#) (void) const  
return true if the edit box has input focus.
- bool [isReadOnly](#) (void) const  
return true if the edit box is read-only.
- size\_t [getCaratIndex](#) (void) const  
return the current position of the carat.
- size\_t [getSelectionStartIndex](#) (void) const  
return the current selection start point.
- size\_t [getSelectionEndIndex](#) (void) const



*return the current selection end point.*

- `size_t` [getSelectionLength](#) (void) const  
*return the length of the current selection (in code points / characters).*
- `size_t` [getMaxTextLength](#) (void) const  
*return the maximum text length set for this edit box.*
- `bool` [isWordWrapped](#) (void) const  
*Return whether the text in the edit box will be word-wrapped.*
- `Scrollbar *` [getVertScrollbar](#) () const  
*Return a pointer to the vertical scrollbar component widget for this [MultiLineEditbox](#).*
- `bool` [isVertScrollbarAlwaysShown](#) (void) const  
*Return whether the vertical scroll bar is always shown.*
- `Scrollbar *` [getHorzScrollbar](#) () const  
*Return a pointer to the horizontal scrollbar component widget for this [MultiLineEditbox](#).*
- `Rect` [getTextRenderArea](#) (void) const  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that the text should be rendered in to.*
- `const` [LineList](#) & [getFormattedLines](#) (void) const
- `size_t` [getLineNumberFromIndex](#) (size\_t index) const  
*Return the line number a given index falls on with the current formatting. Will return last line if index is out of range.*
- `virtual void` [initialiseComponents](#) (void)  
*Initialise the [Window](#) based object ready for use.*
- `void` [setReadOnly](#) (bool setting)  
*Specify whether the edit box is read-only.*
- `void` [setCaratIndex](#) (size\_t carat\_pos)  
*Set the current position of the carat.*
- `void` [setSelection](#) (size\_t start\_pos, size\_t end\_pos)  
*Define the current selection for the edit box.*
- `void` [setMaxTextLength](#) (size\_t max\_len)  
*set the maximum text length for this edit box.*
- `void` [ensureCaratIsVisible](#) (void)  
*Scroll the view so that the current carat position is visible.*
- `void` [setWordWrapping](#) (bool setting)  
*Set whether the text will be word wrapped or not.*

- void `setShowVertScrollbar` (bool setting)  
*Set whether the vertical scroll bar should always be shown.*
- void `setSelectionBrushImage` (const `Image` \*image)
- const `Image` \* `getSelectionBrushImage` () const
- `MultiLineEditbox` (const `String` &type, const `String` &name)  
*Constructor for the `MultiLineEditbox` base class.*
- virtual `~MultiLineEditbox` (void)  
*Destructor for the `MultiLineEditbox` base class.*

## Static Public Attributes

- static const `String` `EventNamespace`  
*Namespace for global events.*
- static const `String` `WidgetTypeName`  
*`Window` factory name.*
- static const `String` `EventReadOnlyModeChanged`  
*The read-only mode for the edit box has been changed.*
- static const `String` `EventWordWrapModeChanged`  
*The word wrap mode of the text box has been changed.*
- static const `String` `EventMaximumTextLengthChanged`  
*The maximum allowable string length has been changed.*
- static const `String` `EventCaratMoved`  
*The text carat (insert point) has changed.*
- static const `String` `EventTextSelectionChanged`  
*The current text selection has changed.*
- static const `String` `EventEditboxFull`  
*The number of characters in the edit box has reached the current maximum.*
- static const `String` `EventVertScrollbarModeChanged`  
*`Event` triggered when the vertical scroll bar 'force' setting changes.*
- static const `String` `EventHorzScrollbarModeChanged`  
*`Event` triggered when the horizontal scroll bar 'force' setting changes.*
- static const `String` `VertScrollbarNameSuffix`  
*Widget name suffix for the vertical scrollbar component.*
- static const `String` `HorzScrollbarNameSuffix`  
*Widget name suffix for the horizontal scrollbar component.*

## Protected Member Functions

- void [formatText](#) (void)  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that the text should be rendered in to.*
- size\_t [getNextTokenLength](#) (const [String](#) &text, size\_t start\_idx) const  
*Return the length of the next token in [String](#) text starting at index start\_idx.*
- void [configureScrollbars](#) (void)  
*display required integrated scroll bars according to current state of the edit box and update their values.*
- size\_t [getTextIndexFromPosition](#) (const [Point](#) &pt) const  
*Return the text code point index that is rendered closest to screen position pt.*
- void [clearSelection](#) (void)  
*Clear the current selection setting.*
- void [eraseSelectedText](#) (bool modify\_text=true)  
*Erase the currently selected text.*
- void [handleBackspace](#) (void)  
*Processing for backspace key.*
- void [handleDelete](#) (void)  
*Processing for Delete key.*
- void [handleCharLeft](#) (uint sysKeys)  
*Processing to move carat one character left.*
- void [handleWordLeft](#) (uint sysKeys)  
*Processing to move carat one word left.*
- void [handleCharRight](#) (uint sysKeys)  
*Processing to move carat one character right.*
- void [handleWordRight](#) (uint sysKeys)  
*Processing to move carat one word right.*
- void [handleDocHome](#) (uint sysKeys)  
*Processing to move carat to the start of the text.*
- void [handleDocEnd](#) (uint sysKeys)  
*Processing to move carat to the end of the text.*
- void [handleLineHome](#) (uint sysKeys)  
*Processing to move carat to the start of the current line.*
- void [handleLineEnd](#) (uint sysKeys)  
*Processing to move carat to the end of the current line.*

- void [handleLineUp](#) (uint sysKeys)  
*Processing to move carat up a line.*
- void [handleLineDown](#) (uint sysKeys)  
*Processing to move carat down a line.*
- void [handleNewLine](#) (uint sysKeys)  
*Processing to insert a new line / paragraph.*
- void [handlePageUp](#) (uint sysKeys)  
*Processing to move caret one page up.*
- void [handlePageDown](#) (uint sysKeys)  
*Processing to move caret one page down.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- bool [handle\\_scrollChange](#) (const [EventArgs](#) &args)  
*Internal handler that is triggered when the user interacts with the scrollbars.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- void [onReadOnlyChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the read-only state of the edit box changes.*
- void [onWordWrapModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the word wrap mode for the the edit box changes.*
- void [onMaximumTextLengthChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the maximum text length for the edit box changes.*
- void [onCaratMoved](#) ([WindowEventArgs](#) &e)  
*Handler called when the carat moves.*
- void [onTextSelectionChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the text selection changes.*
- void [onEditboxFullEvent](#) ([WindowEventArgs](#) &e)  
*Handler called when the edit box is full.*
- void [onVertScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the 'always show' setting for the vertical scroll bar changes.*
- void [onHorzScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when 'always show' setting for the horizontal scroll bar changes.*

- virtual void [onMouseDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void [onMouseDoubleClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been double-clicked within this window's area.*
- virtual void [onMouseTripleClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been triple-clicked within this window's area.*
- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void [onCharacter](#) ([KeyEventArgs](#) &e)  
*Handler called when a character-key has been pressed while this window has input focus.*
- virtual void [onKeyDown](#) ([KeyEventArgs](#) &e)  
*Handler called when a key as been depressed while this window has input focus.*
- virtual void [onTextChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's text is changed.*
- virtual void [onSized](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's size changes.*
- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*

## Protected Attributes

- bool [d\\_readOnly](#)  
*true if the edit box is in read-only mode*
- size\_t [d\\_maxTextLen](#)  
*Maximum number of characters for this *Editbox*.*
- size\_t [d\\_caratPos](#)  
*Position of the carat / insert-point.*
- size\_t [d\\_selectionStart](#)  
*Start of selection area.*
- size\_t [d\\_selectionEnd](#)

*End of selection area.*

- bool [d\\_dragging](#)  
*true when a selection is being dragged.*
- size\_t [d\\_dragAnchorIdx](#)  
*Selection index for drag selection anchor point.*
- bool [d\\_wordWrap](#)  
*true when formatting uses word-wrapping.*
- [LineList](#) [d\\_lines](#)  
*Holds the lines for the current formatting.*
- float [d\\_widestExtent](#)  
*Holds the extent of the widest line as calculated in the last formatting pass.*
- bool [d\\_forceVertScroll](#)  
*true if vertical scrollbar should always be displayed*
- bool [d\\_forceHorzScroll](#)  
*true if horizontal scrollbar should always be displayed*
- const [Image](#) \* [d\\_selectionBrush](#)  
*[Image](#) to use as the selection brush (should be set by derived class).*

## Static Protected Attributes

- static [String](#) [d\\_lineBreakChars](#)  
*Holds what we consider to be line break characters.*

## Classes

- struct [LineInfo](#)  
*struct used to store information about a formatted line within the paragraph.*

### 6.182.1 Detailed Description

Base class for the multi-line edit box widget.

### 6.182.2 Member Function Documentation

#### 6.182.2.1 bool CEGUI::MultiLineEditbox::hasInputFocus (void) const

return true if the edit box has input focus.

**Returns:**

- true if the edit box has keyboard input focus.
- false if the edit box does not have keyboard input focus.

**6.182.2.2 bool CEGUI::MultiLineEditbox::isReadOnly (void) const [inline]**

return true if the edit box is read-only.

**Returns:**

- true if the edit box is read only and can't be edited by the user.
- false if the edit box is not read only and may be edited by the user.

**6.182.2.3 size\_t CEGUI::MultiLineEditbox::getCaratIndex (void) const [inline]**

return the current position of the carat.

**Returns:**

Index of the insert carat relative to the start of the text.

**6.182.2.4 size\_t CEGUI::MultiLineEditbox::getSelectionStartIndex (void) const**

return the current selection start point.

**Returns:**

Index of the selection start point relative to the start of the text. If no selection is defined this function returns the position of the carat.

**6.182.2.5 size\_t CEGUI::MultiLineEditbox::getSelectionEndIndex (void) const**

return the current selection end point.

**Returns:**

Index of the selection end point relative to the start of the text. If no selection is defined this function returns the position of the carat.

**6.182.2.6 size\_t CEGUI::MultiLineEditbox::getSelectionLength (void) const**

return the length of the current selection (in code points / characters).

**Returns:**

Number of code points (or characters) contained within the currently defined selection.

**6.182.2.7** `size_t CEGUI::MultiLineEditbox::getMaxTextLength (void) const` `[inline]`

return the maximum text length set for this edit box.

**Returns:**

The maximum number of code points (characters) that can be entered into this edit box.

**6.182.2.8** `bool CEGUI::MultiLineEditbox::isWordWrapped (void) const`

Return whether the text in the edit box will be word-wrapped.

**Returns:**

- true if the text will be word-wrapped at the edges of the widget frame.
- false if text will not be word-wrapped (a scroll bar will be used to access long text lines).

**6.182.2.9** `Scrollbar * CEGUI::MultiLineEditbox::getVertScrollbar () const`

Return a pointer to the vertical scrollbar component widget for this [MultiLineEditbox](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the vertical [Scrollbar](#) component does not exist.

**6.182.2.10** `bool CEGUI::MultiLineEditbox::isVertScrollbarAlwaysShown (void) const`

Return whether the vertical scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.182.2.11** `Scrollbar * CEGUI::MultiLineEditbox::getHorzScrollbar () const`

Return a pointer to the horizontal scrollbar component widget for this [MultiLineEditbox](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the horizontal [Scrollbar](#) component does not exist.



**6.182.2.12 Rect CEGUI::MultiLineEditbox::getTextRenderArea (void) const**

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that the text should be rendered in to.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering text.

**6.182.2.13 void CEGUI::MultiLineEditbox::initialiseComponents (void) [virtual]**

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.182.2.14 void CEGUI::MultiLineEditbox::setReadOnly (bool *setting*)**

Specify whether the edit box is read-only.

**Parameters:**

- setting* • true if the edit box is read only and can't be edited by the user.
- false if the edit box is not read only and may be edited by the user.

**Returns:**

Nothing.

**6.182.2.15 void CEGUI::MultiLineEditbox::setCaratIndex (size\_t *carat\_pos*)**

Set the current position of the carat.

**Parameters:**

*carat\_pos* New index for the insert carat relative to the start of the text. If the value specified is greater than the number of characters in the edit box, the carat is positioned at the end of the text.

**Returns:**

Nothing.

**6.182.2.16 void CEGUI::MultiLineEditbox::setSelection (size\_t *start\_pos*, size\_t *end\_pos*)**

Define the current selection for the edit box.

**Parameters:**

*start\_pos* Index of the starting point for the selection. If this value is greater than the number of characters in the edit box, the selection start will be set to the end of the text.

*end\_pos* Index of the ending point for the selection. If this value is greater than the number of characters in the edit box, the selection start will be set to the end of the text.

**Returns:**

Nothing.

**6.182.2.17 void CEGUI::MultiLineEditbox::setMaxTextLength (size\_t *max\_len*)**

set the maximum text length for this edit box.

**Parameters:**

*max\_len* The maximum number of code points (characters) that can be entered into this [Editbox](#).

**Returns:**

Nothing.

**6.182.2.18 void CEGUI::MultiLineEditbox::setWordWrapping (bool *setting*)**

Set whether the text will be word wrapped or not.

**Parameters:**

- setting*
- true if the text should word-wrap at the edges of the text box.
  - false if the text should not wrap, but a scroll bar should be used.

**Returns:**

Nothing.

**6.182.2.19 void CEGUI::MultiLineEditbox::setShowVertScrollbar (bool *setting*)**

Set whether the vertical scroll bar should always be shown.

**Parameters:**

*setting* true if the vertical scroll bar should be shown even when it is not required. false if the vertical scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.182.2.20 void CEGUI::MultiLineEditbox::formatText (void)** [protected]

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that the text should be rendered in to.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering text.

Format the text into lines as needed by the current formatting options.

**6.182.2.21 size\_t CEGUI::MultiLineEditbox::getNextTokenLength (const String & text, size\_t start\_idx) const** [protected]

Return the length of the next token in [String](#) *text* starting at index *start\_idx*.

**Note:**

Any single whitespace character is one token, any group of other characters is a token.

**Returns:**

The code point length of the token.

**6.182.2.22 size\_t CEGUI::MultiLineEditbox::getTextIndexFromPosition (const Point & pt) const** [protected]

Return the text code point index that is rendered closest to screen position *pt*.

**Parameters:**

*pt* Point object describing a position on the screen in pixels.

**Returns:**

Code point index into the text that is rendered closest to screen position *pt*.

**6.182.2.23 void CEGUI::MultiLineEditbox::eraseSelectedText (bool modify\_text = true)** [protected]

Erase the currently selected text.

**Parameters:**

*modify\_text* when true, the actual text will be modified. When false, everything is done except erasing the characters.

**6.182.2.24 virtual bool CEGUI::MultiLineEditbox::testClassName\_impl (const String & class\_name) const** [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.182.2.25 virtual bool CEGUI::MultiLineEditbox::validateWindowRenderer (const String & name) const** [inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.182.2.26 void CEGUI::MultiLineEditbox::onMouseButtonDown (MouseEventArgs & e)** [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.182.2.27 void CEGUI::MultiLineEditbox::onMouseButtonUp (MouseEventArgs & e)** [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.182.2.28 void CEGUI::MultiLineEditbox::onMouseDoubleClicked (MouseEventArgs & *e*)**  
[protected, virtual]

Handler called when a mouse button has been double-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.182.2.29 void CEGUI::MultiLineEditbox::onMouseTripleClicked (MouseEventArgs & *e*)**  
[protected, virtual]

Handler called when a mouse button has been triple-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.182.2.30 void CEGUI::MultiLineEditbox::onMouseMove (MouseEventArgs & *e*)**  
[protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.182.2.31 void CEGUI::MultiLineEditbox::onCaptureLost (WindowEventArgs & *e*)**  
[protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.182.2.32 void CEGUI::MultiLineEditbox::onCharacter (KeyEventArgs & *e*)** [protected, virtual]

Handler called when a character-key has been pressed while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'codepoint' field is set to the Unicode code point (encoded as utf32) for the character typed, and whose 'sysKeys' field represents the combination of SystemKey that were active when the event was generated. All other fields should be considered as 'junk'.

Reimplemented from [CEGUI::Window](#).

**6.182.2.33** `void CEGUI::MultiLineEditbox::onKeyDown (KeyEventArgs & e)` [protected, virtual]

Handler called when a key as been depressed while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'scancode' field is set to the Key::Scan value representing the key that was pressed, and whose 'sysKeys' field represents the combination of SystemKey that were active when the event was generated.

Reimplemented from [CEGUI::Window](#).

**6.182.2.34** `void CEGUI::MultiLineEditbox::onTextChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.182.2.35** `void CEGUI::MultiLineEditbox::onSized (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's size changes.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.182.2.36** `void CEGUI::MultiLineEditbox::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

- e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

## 6.183 CEGUI::MultiLineEditbox::LineInfo Struct Reference

struct used to store information about a formatted line within the paragraph.

### Public Attributes

- `size_t d_startIdx`  
*Starting index for this line.*
- `size_t d_length`  
*Code point length of this line.*
- `float d_extent`  
*Rendered extent of this line.*

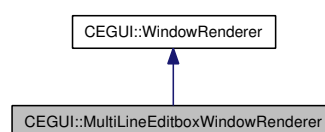
### 6.183.1 Detailed Description

struct used to store information about a formatted line within the paragraph.

## 6.184 CEGUI::MultiLineEditboxWindowRenderer Class Reference

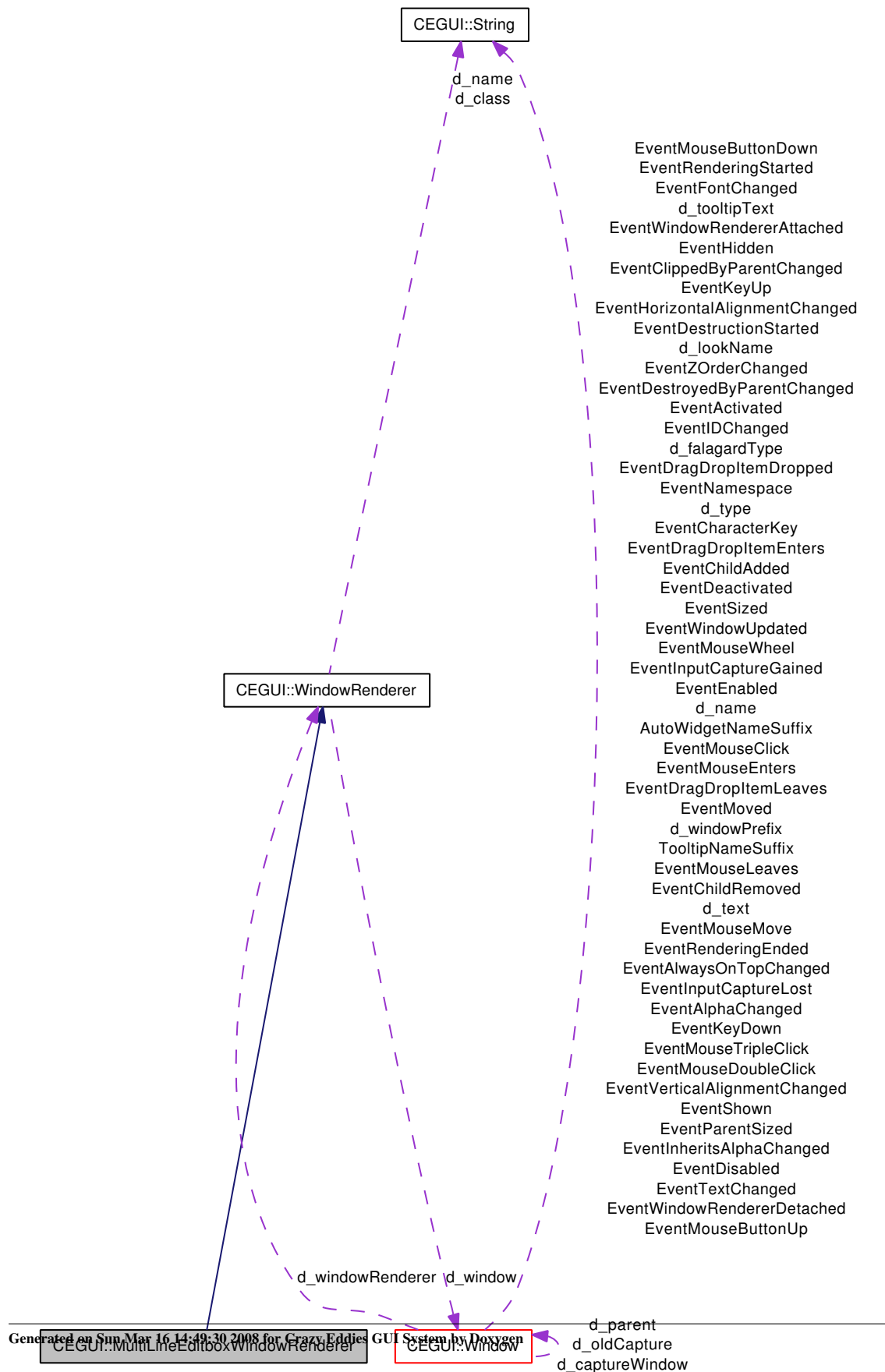
Base class for multi-line edit box window renderer objects.

Inheritance diagram for CEGUI::MultiLineEditboxWindowRenderer:





Collaboration diagram for CEGUI::MultiLineEditboxWindowRenderer:



## Public Member Functions

- [MultiLineEditboxWindowRenderer](#) (const [String](#) &name)

*Constructor.*

- virtual [Rect](#) [getTextRenderArea](#) (void) const =0

*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that the text should be rendered in to.*

### 6.184.1 Detailed Description

Base class for multi-line edit box window renderer objects.

### 6.184.2 Member Function Documentation

#### 6.184.2.1 virtual [Rect](#) CEGUI::MultiLineEditboxWindowRenderer::getTextRenderArea (void) const [pure virtual]

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that the text should be rendered in to.

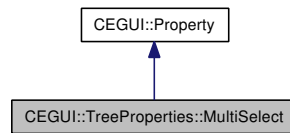
#### Returns:

[Rect](#) object describing the area of the [Window](#) to be used for rendering text.

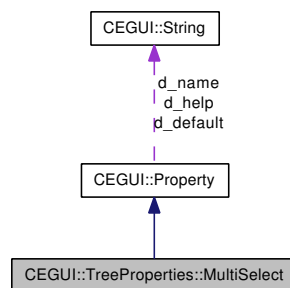
## 6.185 CEGUI::TreeProperties::MultiSelect Class Reference

[Property](#) to access the multi-select setting of the list box.

Inheritance diagram for CEGUI::TreeProperties::MultiSelect:



Collaboration diagram for CEGUI::TreeProperties::MultiSelect:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.185.1 Detailed Description

[Property](#) to access the multi-select setting of the list box.

Usage:

- Name: [MultiSelect](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate that multiple items may be selected.
- "False" to indicate that only a single item may be selected.

## 6.185.2 Member Function Documentation

### 6.185.2.1 `String CEGUI::TreeProperties::MultiSelect::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.185.2.2 `void CEGUI::TreeProperties::MultiSelect::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

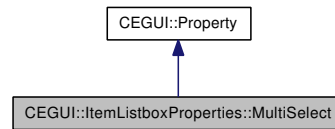
[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

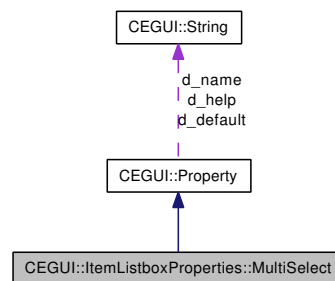
## 6.186 CEGUI::ItemListboxProperties::MultiSelect Class Reference

[Property](#) to access the state of the multiselect enabled setting.

Inheritance diagram for CEGUI::ItemListboxProperties::MultiSelect:



Collaboration diagram for CEGUI::ItemListboxProperties::MultiSelect:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.186.1 Detailed Description

[Property](#) to access the state of the multiselect enabled setting.

**Usage:**

- Name: [MultiSelect](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate that multiselect is enabled.
- "False" to indicate that multiselect is disabled.

## 6.186.2 Member Function Documentation

### 6.186.2.1 `String CEGUI::ItemListboxProperties::MultiSelect::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.186.2.2 `void CEGUI::ItemListboxProperties::MultiSelect::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

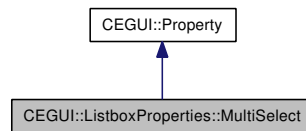
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

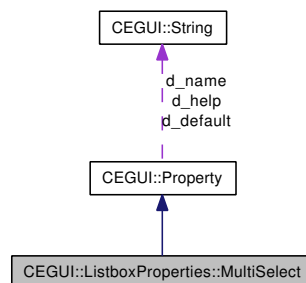
## 6.187 CEGUI::ListboxProperties::MultiSelect Class Reference

[Property](#) to access the multi-select setting of the list box.

Inheritance diagram for CEGUI::ListboxProperties::MultiSelect:



Collaboration diagram for CEGUI::ListboxProperties::MultiSelect:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.187.1 Detailed Description

[Property](#) to access the multi-select setting of the list box.

**Usage:**

- Name: [MultiSelect](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that multiple items may be selected.
- "False" to indicate that only a single item may be selected.

## 6.187.2 Member Function Documentation

### 6.187.2.1 `String CEGUI::ListboxProperties::MultiSelect::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.187.2.2 `void CEGUI::ListboxProperties::MultiSelect::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

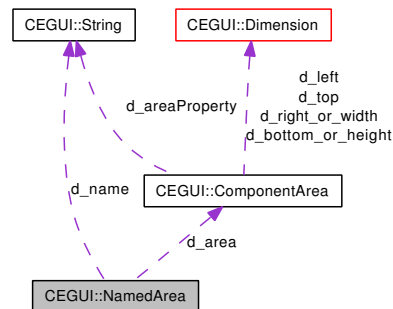
Implements [CEGUI::Property](#).



## 6.188 CEGUI::NamedArea Class Reference

[NamedArea](#) defines an area for a component which may later be obtained and referenced by a name unique to the WidgetLook holding the [NamedArea](#).

Collaboration diagram for CEGUI::NamedArea:



### Public Member Functions

- **NamedArea** (const [String](#) &name)
- const [String](#) & **getName** () const  
*Return the name of this [NamedArea](#).*
- const [ComponentArea](#) & **getArea** () const  
*Return the [ComponentArea](#) of this [NamedArea](#).*
- void **setArea** (const [ComponentArea](#) &area)  
*Set the Area for this [NamedArea](#).*
- void **writeXMLToStream** ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [NamedArea](#) to out\_stream.*

### 6.188.1 Detailed Description

[NamedArea](#) defines an area for a component which may later be obtained and referenced by a name unique to the WidgetLook holding the [NamedArea](#).

### 6.188.2 Member Function Documentation

#### 6.188.2.1 const String & CEGUI::NamedArea::getName () const

Return the name of this [NamedArea](#).

**Returns:**

[String](#) object holding the name of this [NamedArea](#).

**6.188.2.2   const ComponentArea & CEGUI::NamedArea::getArea () const**

Return the [ComponentArea](#) of this [NamedArea](#).

**Returns:**

[ComponentArea](#) object describing the NamedArea's current target area.

**6.188.2.3   void CEGUI::NamedArea::setArea (const ComponentArea & *area*)**

Set the Area for this [NamedArea](#).

**Parameters:**

*area* [ComponentArea](#) object describing a new target area for the [NamedArea](#)..

**Returns:**

Nothing.

**6.188.2.4   void CEGUI::NamedArea::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [NamedArea](#) to *out\_stream*.

**Parameters:**

*out\_stream* Stream where xml data should be output.

*indentLevel* Current XML indentation level

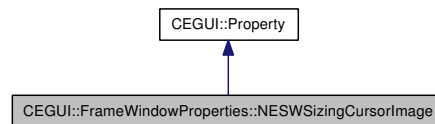
**Returns:**

Nothing.

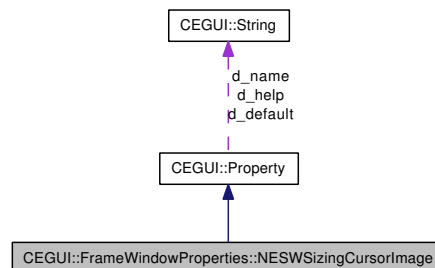
## 6.189 CEGUI::FrameWindowProperties::NESWSizingCursorImage Class Reference

[Property](#) to access the NE-SW diagonal sizing cursor image.

Inheritance diagram for CEGUI::FrameWindowProperties::NESWSizingCursorImage:



Collaboration diagram for CEGUI::FrameWindowProperties::NESWSizingCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.189.1 Detailed Description

[Property](#) to access the NE-SW diagonal sizing cursor image.

Usage:

- Name: [NESWSizingCursorImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.189.2 Member Function Documentation

#### 6.189.2.1 String CEGUI::FrameWindowProperties::NESWSizingCursorImage::get (const [PropertyReceiver](#) \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.189.2.2** `void CEGUI::FrameWindowProperties::NESWSizingCursorImage::set  
(PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

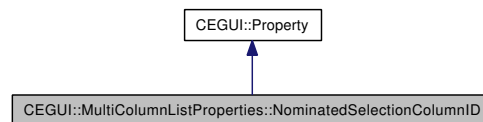
[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

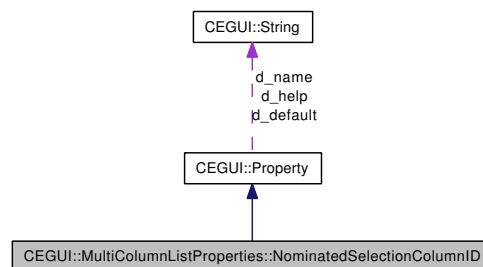
## 6.190 CEGUI::MultiColumnListProperties::NominatedSelectionColumnID Class Reference

[Property](#) to access the nominated selection column (via ID).

Inheritance diagram for CEGUI::MultiColumnListProperties::NominatedSelectionColumnID:



Collaboration diagram for CEGUI::MultiColumnListProperties::NominatedSelectionColumnID:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.190.1 Detailed Description

[Property](#) to access the nominated selection column (via ID).

#### Usage:

- Name: [NominatedSelectionColumnID](#)
- Format: "[uint]".

#### Where:

- [uint] is any unsigned integer value representing the ID code of the column to be used.

## 6.190.2 Member Function Documentation

### 6.190.2.1 String CEGUI::MultiColumnListProperties::NominatedSelectionColumnID::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.190.2.2 void CEGUI::MultiColumnListProperties::NominatedSelectionColumnID::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

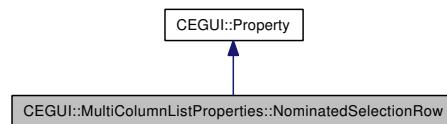
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

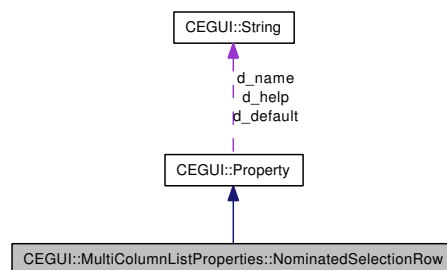
## 6.191 CEGUI::MultiColumnListProperties::NominatedSelectionRow Class Reference

[Property](#) to access the nominated selection row.

Inheritance diagram for CEGUI::MultiColumnListProperties::NominatedSelectionRow:



Collaboration diagram for CEGUI::MultiColumnListProperties::NominatedSelectionRow:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.191.1 Detailed Description

[Property](#) to access the nominated selection row.

##### Usage:

- Name: [NominatedSelectionRow](#)
- Format: "[uint]".

##### Where:

- [uint] is any unsigned integer value representing the index of the row to be used.

## 6.191.2 Member Function Documentation

### 6.191.2.1 String CEGUI::MultiColumnListProperties::NominatedSelectionRow::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.191.2.2 void CEGUI::MultiColumnListProperties::NominatedSelectionRow::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

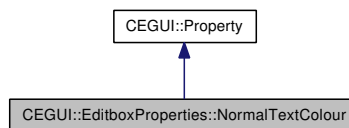
Implements [CEGUI::Property](#).



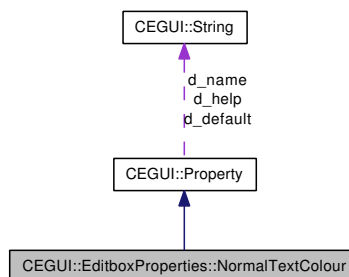
## 6.192 CEGUI::EditboxProperties::NormalTextColour Class Reference

[Property](#) to access the normal, unselected, text [colour](#) used for rendering text.

Inheritance diagram for CEGUI::EditboxProperties::NormalTextColour:



Collaboration diagram for CEGUI::EditboxProperties::NormalTextColour:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.192.1 Detailed Description

[Property](#) to access the normal, unselected, text [colour](#) used for rendering text.

##### Usage:

- Name: [NormalTextColour](#)
- Format: "aarrggbb".

##### Where:

- aarrggbb is the ARGB [colour](#) value to be used.

## 6.192.2 Member Function Documentation

### 6.192.2.1 String CEGUI::EditboxProperties::NormalTextColour::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.192.2.2 void CEGUI::EditboxProperties::NormalTextColour::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

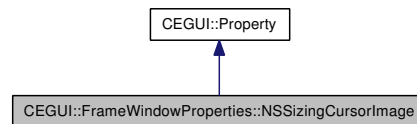
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

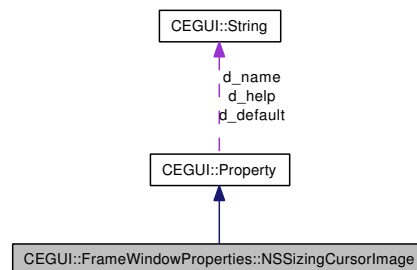
## 6.193 CEGUI::FrameWindowProperties::NSSizingCursorImage Class Reference

[Property](#) to access the N-S (up-down) sizing cursor image.

Inheritance diagram for CEGUI::FrameWindowProperties::NSSizingCursorImage:



Collaboration diagram for CEGUI::FrameWindowProperties::NSSizingCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

### 6.193.1 Detailed Description

[Property](#) to access the N-S (up-down) sizing cursor image.

Usage:

- Name: [NSSizingCursorImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.193.2 Member Function Documentation

#### 6.193.2.1 String CEGUI::FrameWindowProperties::NSSizingCursorImage::get (const [PropertyReceiver](#) \* receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.193.2.2** `void CEGUI::FrameWindowProperties::NSSizingCursorImage::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

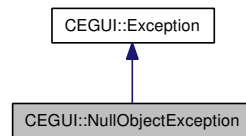
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

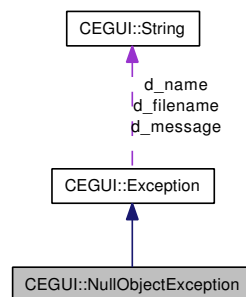
## 6.194 CEGUI::NullObjectException Class Reference

[Exception](#) class used when some required object or parameter is null.

Inheritance diagram for CEGUI::NullObjectException:



Collaboration diagram for CEGUI::NullObjectException:



### Public Member Functions

- [NullObjectException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the null object exception by calling the base class.*

#### 6.194.1 Detailed Description

[Exception](#) class used when some required object or parameter is null.

#### 6.194.2 Constructor & Destructor Documentation

- 6.194.2.1 CEGUI::NullObjectException::NullObjectException (const [String](#) & message, const [String](#) &file = "unknown", int line = 0) [inline]**

Constructor that is responsible for logging the null object exception by calling the base class.

##### Parameters:

- message* [String](#) object describing the reason for the null object exception being thrown.
- filename* [String](#) object containing the name of the file where the null object exception occurred.
- line* Integer representing the line number where the null object exception occurred.

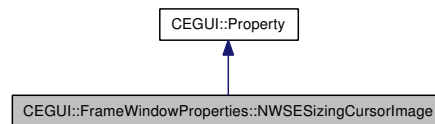
**Remarks:**

The null object exception name is automatically passed to the base class as "CEGUI::NullObjectException".

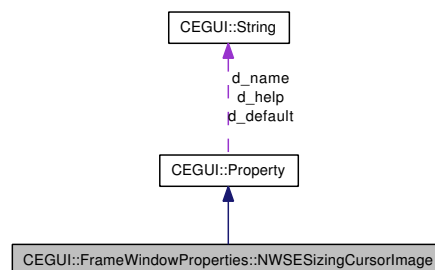
## 6.195 CEGUI::FrameWindowProperties::NWSEizingCursorImage Class Reference

[Property](#) to access the NW-SE diagonal sizing cursor image.

Inheritance diagram for CEGUI::FrameWindowProperties::NWSEizingCursorImage:



Collaboration diagram for CEGUI::FrameWindowProperties::NWSEizingCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.195.1 Detailed Description

[Property](#) to access the NW-SE diagonal sizing cursor image.

Usage:

- Name: [NWSEizingCursorImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.195.2 Member Function Documentation

#### 6.195.2.1 String CEGUI::FrameWindowProperties::NWSEizingCursorImage::get (const [PropertyReceiver](#) \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.195.2.2** `void CEGUI::FrameWindowProperties::NWSEsizingCursorImage::set  
(PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

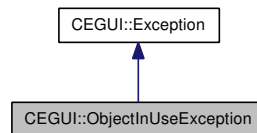
Implements [CEGUI::Property](#).



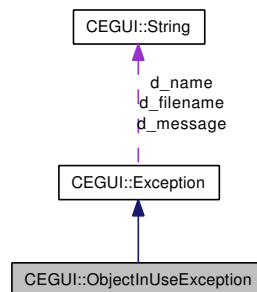
## 6.196 CEGUI::ObjectInUseException Class Reference

**Exception** class used when some attempt to delete, remove, or otherwise invalidate some object that is still in use occurs.

Inheritance diagram for CEGUI::ObjectInUseException:



Collaboration diagram for CEGUI::ObjectInUseException:



### Public Member Functions

- **ObjectInUseException** (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the object in use exception by calling the base class.*

### 6.196.1 Detailed Description

**Exception** class used when some attempt to delete, remove, or otherwise invalidate some object that is still in use occurs.

### 6.196.2 Constructor & Destructor Documentation

#### 6.196.2.1 CEGUI::ObjectInUseException::ObjectInUseException (const [String](#) & message, const [String](#) &file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the object in use exception by calling the base class.

#### Parameters:

- message** [String](#) object describing the reason for the object in use exception being thrown.
- filename** [String](#) object containing the name of the file where the object in use exception occurred.
- line** Integer representing the line number where the object in use exception occurred.

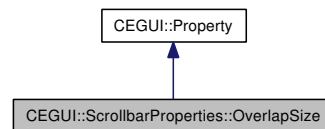
**Remarks:**

The object in use exception name is automatically passed to the base class as "CEGUI::ObjectInUseException".

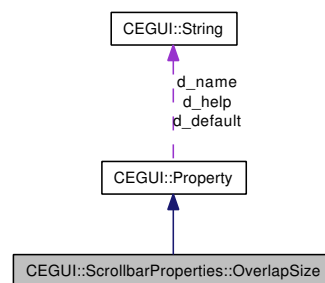
## 6.197 CEGUI::ScrollbarProperties::OverlapSize Class Reference

[Property](#) to access the overlap size for the [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollbarProperties::OverlapSize:



Collaboration diagram for CEGUI::ScrollbarProperties::OverlapSize:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.197.1 Detailed Description

[Property](#) to access the overlap size for the [Scrollbar](#).

#### Usage:

- Name: [OverlapSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the per-page overlap (as defined by the client code).

## 6.197.2 Member Function Documentation

### 6.197.2.1 `String CEGUI::ScrollbarProperties::OverlapSize::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.197.2.2 `void CEGUI::ScrollbarProperties::OverlapSize::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

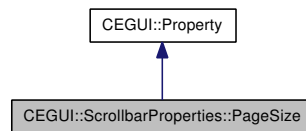
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

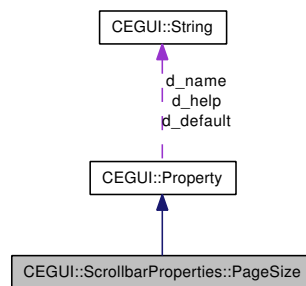
## 6.198 CEGUI::ScrollbarProperties::PageSize Class Reference

[Property](#) to access the page size for the [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollbarProperties::PageSize:



Collaboration diagram for CEGUI::ScrollbarProperties::PageSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.198.1 Detailed Description

[Property](#) to access the page size for the [Scrollbar](#).

#### Usage:

- Name: [PageSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the visible page (as defined by the client code).

## 6.198.2 Member Function Documentation

### 6.198.2.1 `String CEGUI::ScrollbarProperties::PageSize::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.198.2.2 `void CEGUI::ScrollbarProperties::PageSize::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

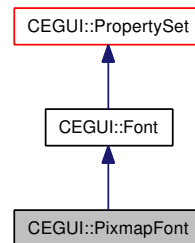
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

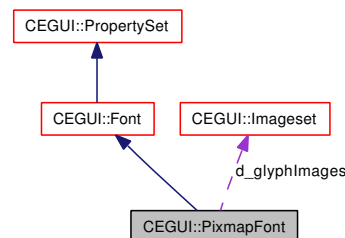
## 6.199 CEGUI::PixmapFont Class Reference

Implementation of the [Font](#) class interface using static Imageset's.

Inheritance diagram for CEGUI::PixmapFont:



Collaboration diagram for CEGUI::PixmapFont:



### Public Member Functions

- virtual void [load](#) ()

Complete font loading. If you create the font from an XML file, this method is invoked automatically after reading all the required data from the [XMLAttributes](#) object. If you create the font manually, it is your responsibility to call this function as soon as you set up all the appropriate fields of the [Font](#) object.

### Protected Member Functions

- [PixmapFont](#) (const [String](#) &name, const [String](#) &fontname, const [String](#) &resourceGroup="")

Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to define the glyph mapping (via the Mapping property), and finally call the [load\(\)](#) method which will make font available for use (this is not *\*required\** for [PixmapFont](#), though).

- [PixmapFont](#) (const [XMLAttributes](#) &attributes)

Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to set up all remaining font parameters after constructing the [Font](#) object, and finally calling the [load\(\)](#) method which will make font available for use. All font parameters that are not initialized are set to sensible default values.

**Parameters:**

**name** The unique name that will be used to identify this [Font](#).

**fontname** The filename of the font file, which contains the font data. This can be a TrueType, PostScript, bitmap font etc file.

**resourceGroup** Resource group identifier to be passed to the resource provider to load the font definition file.

- virtual [~PixmapFont](#) ()  
*Destroys a [Font](#) object.*
- virtual void [updateFont](#) ()  
*Update the font as required according to the current parameters.*
- virtual void [writeXMLToStream\\_impl](#) (XMLSerializer &xml\_stream) const  
*Same as [writeXMLToStream\(\)](#) but called from inside [writeXMLToStream\(\)](#) so that derived classes may add their own attributes to stream.*  
**Parameters:**  
***xml\_stream** Stream where xml data should be output.*
- virtual void [defineMapping](#) (const XMLAttributes &attributes)  
*Define a glyph mapping (handle a <Mapping> XML element).*
- void [defineMapping](#) (String image\_name, utf32 codepoint, float horzAdvance)  
*Define a single glyph mapping.*
- void [addPixmapFontProperties](#) ()  
*Register all properties of this class.*
- void [reinit](#) ()  
*Initialize the imageset.*

## Protected Attributes

- [Imageset](#) \* [d\\_glyphImages](#)  
*The imageset with the glyphs.*
- float [d\\_origHorzScaling](#)  
*Current X scaling for glyph images.*
- bool [d\\_imagesetOwner](#)  
*true if we're the owners of the imageset*

## Friends

- class **FontManager**
- class **FontProperties::PixmapImageset**
- class **FontProperties::PixmapMapping**

### 6.199.1 Detailed Description

Implementation of the [Font](#) class interface using static Imageset's.

To create such a font you must create a [Imageset](#) with all the glyphs, and then define individual glyphs via the Mapping object property (or via the Mapping XML element).



## 6.199.2 Constructor & Destructor Documentation

### 6.199.2.1 CEGUI::PixmapFont::PixmapFont (const String & *name*, const String & *fontname*, const String & *resourceGroup* = "") [protected]

Constructs a new semi-complete [Font](#) object. It is the responsibility of the user to define the glyph mapping (via the Mapping property), and finally call the [load\(\)](#) method which will make font available for use (this is not *\*required\** for [PixmapFont](#), though).

All font parameters that are not initialized are set to sensible default values.

#### Parameters:

***name*** The unique name that will be used to identify this [Font](#).

***fontname*** The filename of the font file, which contains the font data. This can be a TrueType, PostScript, bitmap font etc file. If resourceGroup is set to the special value of "\*", fontname is interpreted as a imageset name and the respective [Imageset](#) object must be already loaded.

***resourceGroup*** Resource group identifier to be passed to the resource provider to load the font definition file.

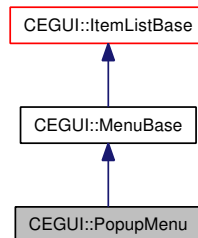
#### Exceptions:

**[UnknownObjectException](#)** Thrown if no [Imageset](#) named *filename* is present in within the system (when resourceGroup == "\*").

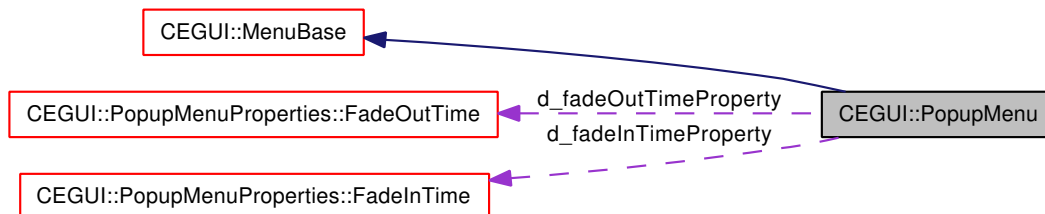
## 6.200 CEGUI::PopupMenu Class Reference

Base class for popup menus.

Inheritance diagram for CEGUI::PopupMenu:



Collaboration diagram for CEGUI::PopupMenu:



### Public Member Functions

- float [getFadeInTime](#) (void) const  
*Get the fade in time for this popup menu.*
- float [getFadeOutTime](#) (void) const  
*Get the fade out time for this popup menu.*
- bool [isPopupMenuOpen](#) (void) const  
*Find out if this popup menu is open or closed;.*
- void [setFadeInTime](#) (float fadetime)  
*Set the fade in time for this popup menu.*
- void [setFadeOutTime](#) (float fadetime)  
*Set the fade out time for this popup menu.*
- void [openPopupMenu](#) (bool notify=true)  
*Tells the popup menu to open.*
- void [closePopupMenu](#) (bool notify=true)  
*Tells the popup menu to close.*
- [PopupMenu](#) (const [String](#) &type, const [String](#) &name)

Constructor for *PopupMenu* objects.

- virtual `~PopupMenu` (void)

Destructor for *PopupMenu* objects.

## Static Public Attributes

- static const `String EventNamespace`

Namespace for global events.

- static const `String WidgetTypeName`

*Window* factory name.

## Protected Member Functions

- virtual void `updateSelf` (float elapsed)

Perform actual update processing for this *Window*.

- virtual void `layoutItemWidgets` (void)

Setup size and position for the item widgets attached to this *Listbox*.

- virtual `Size getContentSize` (void) const

Resizes the popup menu to exactly fit the content that is attached to it.

- virtual bool `testClassName_impl` (const `String &class_name`) const

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

- virtual void `onAlphaChanged` (`WindowEventArgs &e`)

Handler called when the window's alpha blend value is changed.

- virtual void `onDestructionStarted` (`WindowEventArgs &e`)

Handler called when this window's destruction sequence has begun.

- virtual void `onShown` (`WindowEventArgs &e`)

Handler called when the window is shown (made visible).

- virtual void `onHidden` (`WindowEventArgs &e`)

Handler called when the window is hidden.

- virtual void `onMouseButtonDown` (`MouseEventArgs &e`)

Handler called when a mouse button has been depressed within this window's area.

- virtual void `onMouseButtonUp` (`MouseEventArgs &e`)

Handler called when a mouse button has been released within this window's area.

## Protected Attributes

- float [d\\_origAlpha](#)  
*The original alpha of this window.*
- float [d\\_fadeElapsed](#)  
*The time in seconds this popup menu has been fading.*
- float [d\\_fadeOutTime](#)  
*The time in seconds it takes for this popup menu to fade out.*
- float [d\\_fadeInTime](#)  
*The time in seconds it takes for this popup menu to fade in.*
- bool [d\\_fading](#)  
*true if this popup menu is fading in/out. false if not*
- bool [d\\_fadingOut](#)  
*true if this popup menu is fading out. false if fading in.*
- bool [d\\_isOpen](#)  
*true if this popup menu is open. false if not.*

### 6.200.1 Detailed Description

Base class for popup menus.

### 6.200.2 Member Function Documentation

#### 6.200.2.1 float CEGUI::PopupMenu::getFadeInTime (void) const [inline]

Get the fade in time for this popup menu.

**Returns:**

The time in seconds that it takes for the popup to fade in. 0 if fading is disabled.

#### 6.200.2.2 float CEGUI::PopupMenu::getFadeOutTime (void) const [inline]

Get the fade out time for this popup menu.

**Returns:**

The time in seconds that it takes for the popup to fade out. 0 if fading is disabled.

**6.200.2.3 void CEGUI::PopupMenu::setFadeInTime (float *fadetime*)** [inline]

Set the fade in time for this popup menu.

**Parameters:**

*fadetime* The time in seconds that it takes for the popup to fade in. If this parameter is zero, fading is disabled.

**6.200.2.4 void CEGUI::PopupMenu::setFadeOutTime (float *fadetime*)** [inline]

Set the fade out time for this popup menu.

**Parameters:**

*fadetime* The time in seconds that it takes for the popup to fade out. If this parameter is zero, fading is disabled.

**6.200.2.5 void CEGUI::PopupMenu::openPopupMenu (bool *notify* = true)**

Tells the popup menu to open.

**Parameters:**

*notify* true if the parent menu item (if any) is to handle the opening. false if not.

**6.200.2.6 void CEGUI::PopupMenu::closePopupMenu (bool *notify* = true)**

Tells the popup menu to close.

**Parameters:**

*notify* true if the parent menu item (if any) is to handle the closing. false if not.

**6.200.2.7 void CEGUI::PopupMenu::updateSelf (float *elapsed*)** [protected, virtual]

Perform actual update processing for this [Window](#).

**Parameters:**

*elapsed* float value indicating the number of seconds elapsed since the last update call.

**Returns:**

Nothing.

Reimplemented from [CEGUI::Window](#).

**6.200.2.8 void CEGUI::PopupMenu::layoutItemWidgets (void) [protected, virtual]**

Setup size and position for the item widgets attached to this [Listbox](#).

**Returns:**

Nothing.

Implements [CEGUI::ItemListBase](#).

**6.200.2.9 Size CEGUI::PopupMenu::getContentSize (void) const [protected, virtual]**

Resizes the popup menu to exactly fit the content that is attached to it.

**Returns:**

Nothing.

Implements [CEGUI::ItemListBase](#).

**6.200.2.10 virtual bool CEGUI::PopupMenu::testClassName\_impl (const String & class\_name) const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::MenuBase](#).

**6.200.2.11 void CEGUI::PopupMenu::onAlphaChanged (WindowEventArgs & e) [protected, virtual]**

Handler called when the window's alpha blend value is changed.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.200.2.12 void CEGUI::PopupMenu::onDestructionStarted (WindowEventArgs & e) [protected, virtual]**

Handler called when this window's destruction sequence has begun.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.200.2.13** `void CEGUI::PopupMenu::onShown (WindowEventArgs & e)` [protected, virtual]

Handler called when the window is shown (made visible).

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.200.2.14** `void CEGUI::PopupMenu::onHidden (WindowEventArgs & e)` [protected, virtual]

Handler called when the window is hidden.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.200.2.15** `void CEGUI::PopupMenu::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

- e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.200.2.16** `void CEGUI::PopupMenu::onMouseButtonUp (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

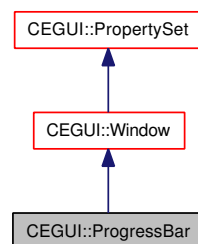
- e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

## 6.201 CEGUI::ProgressBar Class Reference

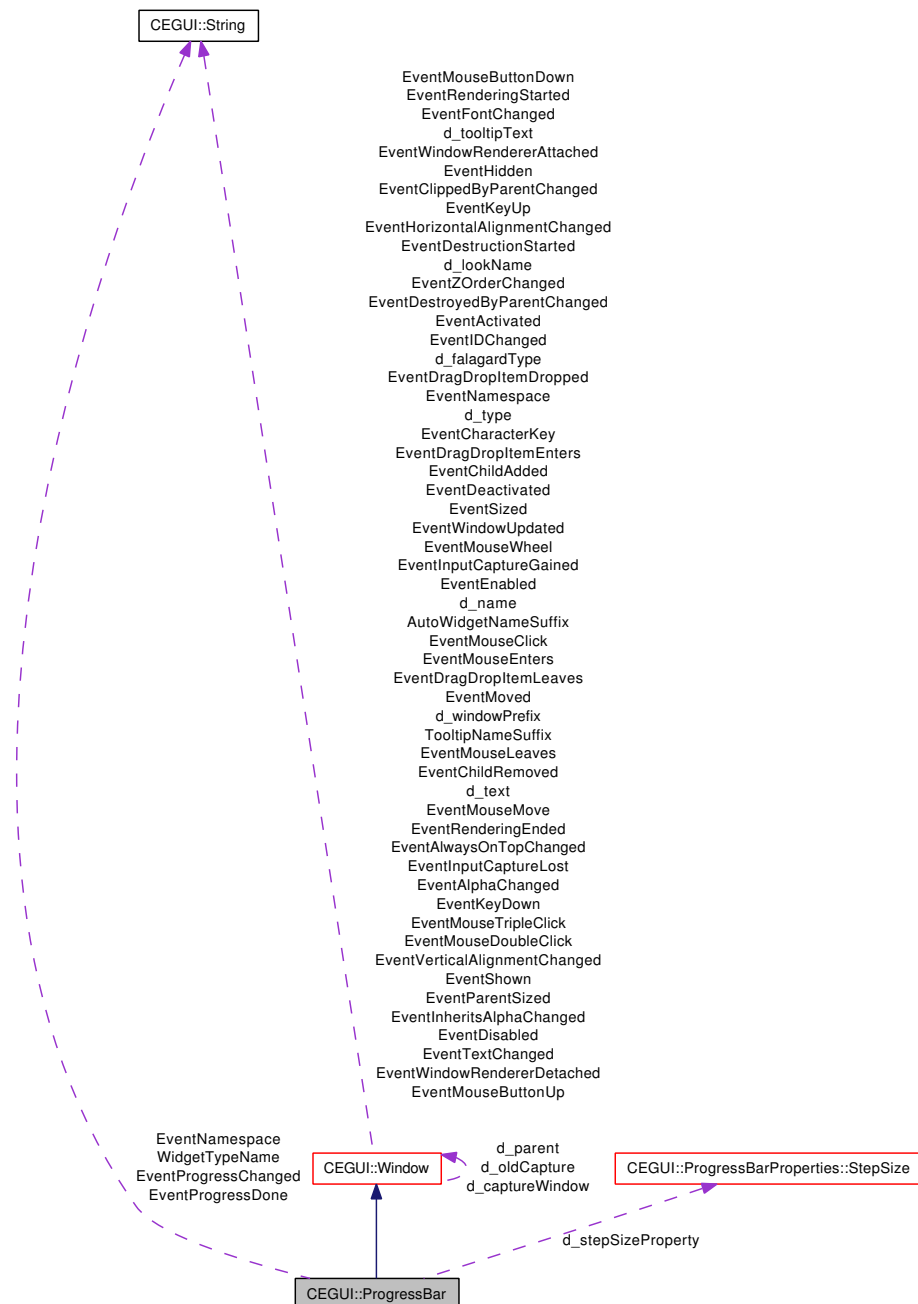
Base class for progress bars.

Inheritance diagram for CEGUI::ProgressBar:





Collaboration diagram for CEGUI::ProgressBar:



## Public Member Functions

- float `getProgress` (void) const  
*return the current progress value*
- float `getStep` (void) const  
*return the current step size*

- void [setProgress](#) (float progress)  
*set the current progress.*
- void [setStepSize](#) (float step\_val)  
*set the size of the 'step' in percentage points (default is 0.01f or 1%).*
- void [step](#) (void)  
*cause the progress to step*
- void [adjustProgress](#) (float delta)  
*Modify the progress level by a specified delta.*
- [ProgressBar](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [ProgressBar](#) class.*
- virtual [~ProgressBar](#) (void)  
*Destructor for [ProgressBar](#).*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventProgressChanged](#)  
*[Event](#) fired whenever the progress value changes.*
- static const [String](#) [EventProgressDone](#)  
*[Event](#) fired when the progress bar reaches 100%.*

## Protected Member Functions

- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [onProgressChanged](#) ([WindowEventArgs](#) &e)  
*event triggered when progress changes*
- virtual void [onProgressDone](#) ([WindowEventArgs](#) &e)  
*event triggered when progress reaches 100%*

## Protected Attributes

- float `d_progress`  
*current progress (from 0.0f to 1.0f)*
- float `d_step`  
*amount to 'step' progress by on a call to `step()`*

### 6.201.1 Detailed Description

Base class for progress bars.

### 6.201.2 Member Function Documentation

#### 6.201.2.1 void CEGUI::ProgressBar::setProgress (float *progress*)

set the current progress.

##### Parameters:

*progress* The level of progress to set. If this value is >1.0f (100%) progress will be limited to 1.0f.

##### Returns:

Nothing.

#### 6.201.2.2 void CEGUI::ProgressBar::setStepSize (float *step\_val*) [inline]

set the size of the 'step' in percentage points (default is 0.01f or 1%).

##### Parameters:

*step* Amount to increase the progress by each time the step method is called.

##### Returns:

Nothing.

#### 6.201.2.3 void CEGUI::ProgressBar::step (void) [inline]

cause the progress to step

The amount the progress bar will step can be changed by calling the setStepSize method. The default step size is 0.01f which is equal to 1%.

##### Returns:

Nothing.

**6.201.2.4 void CEGUI::ProgressBar::adjustProgress (float *delta*) [inline]**

Modify the progress level by a specified delta.

**Parameters:**

*delta* amount to adjust the progress by. Whatever this value is, the progress of the bar will be kept within the range:  $0.0f \leq \text{progress} \leq 1.0f$ .

**Returns:**

Nothing.

**6.201.2.5 virtual bool CEGUI::ProgressBar::testClassName\_impl (const String & *class\_name*) const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

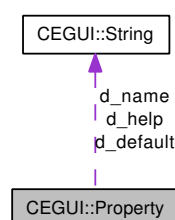
## 6.202 CEGUI::Property Class Reference

An abstract class that defines the interface to access object properties by name.

Inherited by CEGUI::CheckboxProperties::Selected, CEGUI::ComboboxProperties::CaratIndex, CEGUI::ComboboxProperties::EditSelectionLength, CEGUI::ComboboxProperties::EditSelectionStart, CEGUI::ComboboxProperties::ForceHorzScrollbar, CEGUI::ComboboxProperties::ForceVertScrollbar, CEGUI::ComboboxProperties::MaxEditTextLength, CEGUI::ComboboxProperties::ReadOnly, CEGUI::ComboboxProperties::SingleClickMode, CEGUI::ComboboxProperties::SortList, CEGUI::ComboboxProperties::ValidationString, CEGUI::DragContainerProperties::DragAlpha, CEGUI::DragContainerProperties::DragCursorImage, CEGUI::DragContainerProperties::DraggingEnabled, CEGUI::DragContainerProperties::DragThreshold, CEGUI::EditboxProperties::ActiveSelectionColour, CEGUI::EditboxProperties::CaratIndex, CEGUI::EditboxProperties::InactiveSelectionColour, CEGUI::EditboxProperties::MaskCodepoint, CEGUI::EditboxProperties::MaskText, CEGUI::EditboxProperties::MaxTextLength, CEGUI::EditboxProperties::NormalTextColour, CEGUI::EditboxProperties::ReadOnly, CEGUI::EditboxProperties::SelectedTextColour, CEGUI::EditboxProperties::SelectionLength, CEGUI::EditboxProperties::SelectionStart, CEGUI::EditboxProperties::ValidationString, CEGUI::FontProperties::AutoScaled, CEGUI::FontProperties::FileName, CEGUI::FontProperties::FreeTypeAntialiased, CEGUI::FontProperties::FreeTypePointSize, CEGUI::FontProperties::Name, CEGUI::FontProperties::NativeRes, CEGUI::FontProperties::PixmapImageset, CEGUI::FontProperties::PixmapMapping, CEGUI::FontProperties::ResourceGroup, CEGUI::FrameWindowProperties::CloseButtonEnabled, CEGUI::FrameWindowProperties::DragMovingEnabled, CEGUI::FrameWindowProperties::EWSizingCursorImage, CEGUI::FrameWindowProperties::FrameEnabled, CEGUI::FrameWindowProperties::NESWSizingCursorImage, CEGUI::FrameWindowProperties::NSSizingCursorImage, CEGUI::FrameWindowProperties::NWSESizingCursorImage, CEGUI::FrameWindowProperties::RollUpEnabled, CEGUI::FrameWindowProperties::RollUpState, CEGUI::FrameWindowProperties::SizingBorderThickness, CEGUI::FrameWindowProperties::SizingEnabled, CEGUI::FrameWindowProperties::TitlebarEnabled, CEGUI::ItemEntryProperties::Selectable, CEGUI::ItemEntryProperties::Selected, CEGUI::ItemListBaseProperties::AutoResizeEnabled, CEGUI::ItemListBaseProperties::SortEnabled, CEGUI::ItemListBaseProperties::SortMode, CEGUI::ItemListBoxProperties::MultiSelect, CEGUI::ListboxProperties::ForceHorzScrollbar, CEGUI::ListboxProperties::ForceVertScrollbar, CEGUI::ListboxProperties::ItemTooltips, CEGUI::ListboxProperties::MultiSelect, CEGUI::ListboxProperties::Sort, CEGUI::ListHeaderProperties::ColumnsMovable, CEGUI::ListHeaderProperties::ColumnsSizable, CEGUI::ListHeaderProperties::SortColumnID, CEGUI::ListHeaderProperties::SortDirection, CEGUI::ListHeaderProperties::SortSettingEnabled, CEGUI::ListHeaderSegmentProperties::Clickable, CEGUI::ListHeaderSegmentProperties::Dragable, CEGUI::ListHeaderSegmentProperties::MovingCursorImage, CEGUI::ListHeaderSegmentProperties::Sizable, CEGUI::ListHeaderSegmentProperties::SizingCursorImage, CEGUI::ListHeaderSegmentProperties::SortDirection, CEGUI::MenuBaseProperties::AllowMultiplePopups, CEGUI::MenuBaseProperties::ItemSpacing, CEGUI::MultiColumnListProperties::ColumnHeader, CEGUI::MultiColumnListProperties::ColumnsMovable, CEGUI::MultiColumnListProperties::ColumnsSizable, CEGUI::MultiColumnListProperties::ForceHorzScrollbar, CEGUI::MultiColumnListProperties::ForceVertScrollbar, CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, CEGUI::MultiColumnListProperties::NominatedSelectionRow, CEGUI::MultiColumnListProperties::RowCount, CEGUI::MultiColumnListProperties::SelectionMode, CEGUI::MultiColumnListProperties::SortColumnID, CEGUI::MultiColumnListProperties::SortDirection, CEGUI::MultiColumnListProperties::SortSettingEnabled, CEGUI::MultiLineEditboxProperties::CaratIndex, CEGUI::MultiLineEditboxProperties::ForceVertScrollbar, CEGUI::MultiLineEditboxProperties::MaxTextLength, CEGUI::MultiLineEditboxProperties::ReadOnly, CEGUI::MultiLineEditboxProperties::SelectionBrushImage, CEGUI::MultiLineEditboxProperties::SelectionLength, CEGUI::MultiLineEditboxProperties::SelectionStart, CEGUI::MultiLineEditboxProperties::WordWrap, CEGUI::PopupMenuProperties::FadeInTime, CEGUI::PopupMenuProperties::FadeOutTime, CEGUI::ProgressBarProperties::CurrentProgress, CEGUI::ProgressBarProperties::StepSize, CEGUI::PropertyDefinitionBase, CEGUI::RadioButtonProperties::GroupID, CEGUI::RadioButtonProperties::Selected, CEGUI::ScrollablePaneProperties::ContentArea,

CEGUI::ScrollablePaneProperties::ContentPaneAutoSized, CEGUI::ScrollablePaneProperties::ForceHorzScrollbar,  
 CEGUI::ScrollablePaneProperties::ForceVertScrollbar, CEGUI::ScrollablePaneProperties::HorzOverlapSize,  
 CEGUI::ScrollablePaneProperties::HorzScrollPosition, CEGUI::ScrollablePaneProperties::HorzStepSize,  
 CEGUI::ScrollablePaneProperties::VertOverlapSize, CEGUI::ScrollablePaneProperties::VertScrollPosition,  
 CEGUI::ScrollablePaneProperties::VertStepSize, CEGUI::ScrollbarProperties::DocumentSize,  
 CEGUI::ScrollbarProperties::OverlapSize, CEGUI::ScrollbarProperties::PageSize,  
 CEGUI::ScrollbarProperties::ScrollPosition, CEGUI::ScrollbarProperties::StepSize,  
 CEGUI::ScrolledContainerProperties::ChildExtentsArea, CEGUI::ScrolledContainerProperties::ContentArea,  
 CEGUI::ScrolledContainerProperties::ContentPaneAutoSized, CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar,  
 CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar, CEGUI::SliderProperties::ClickStepSize,  
 CEGUI::SliderProperties::CurrentValue, CEGUI::SliderProperties::MaximumValue,  
 CEGUI::SpinnerProperties::CurrentValue, CEGUI::SpinnerProperties::MaximumValue,  
 CEGUI::SpinnerProperties::MinimumValue, CEGUI::SpinnerProperties::StepSize,  
 CEGUI::SpinnerProperties::TextInputMode, CEGUI::TabControlProperties::TabHeight,  
 CEGUI::TabControlProperties::TabPagePosition, CEGUI::TabControlProperties::TabTextPadding,  
 CEGUI::ThumbProperties::HorzFree, CEGUI::ThumbProperties::HorzRange,  
 CEGUI::ThumbProperties::HotTracked, CEGUI::ThumbProperties::VertFree,  
 CEGUI::ThumbProperties::VertRange, CEGUI::TitlebarProperties::DraggingEnabled,  
 CEGUI::TooltipProperties::DisplayTime, CEGUI::TooltipProperties::FadeTime,  
 CEGUI::TooltipProperties::HoverTime, CEGUI::TreeProperties::ForceHorzScrollbar,  
 CEGUI::TreeProperties::ForceVertScrollbar, CEGUI::TreeProperties::ItemTooltips,  
 CEGUI::TreeProperties::MultiSelect, CEGUI::TreeProperties::Sort, CEGUI::WindowProperties::Alpha,  
 CEGUI::WindowProperties::AlwaysOnTop, CEGUI::WindowProperties::AutoRepeatDelay,  
 CEGUI::WindowProperties::AutoRepeatRate, CEGUI::WindowProperties::ClippedByParent,  
 CEGUI::WindowProperties::CustomTooltipType, CEGUI::WindowProperties::DestroyedByParent,  
 CEGUI::WindowProperties::Disabled, CEGUI::WindowProperties::DistributeCapturedInputs,  
 CEGUI::WindowProperties::DragDropTarget, CEGUI::WindowProperties::Font,  
 CEGUI::WindowProperties::HorizontalAlignment, CEGUI::WindowProperties::ID,  
 CEGUI::WindowProperties::InheritsAlpha, CEGUI::WindowProperties::InheritsTooltipText,  
 CEGUI::WindowProperties::LookNFeel, CEGUI::WindowProperties::MouseButtonDownAutoRepeat,  
 CEGUI::WindowProperties::MouseCursorImage, CEGUI::WindowProperties::MousePassThroughEnabled,  
 CEGUI::WindowProperties::RestoreOldCapture, CEGUI::WindowProperties::RiseOnClick,  
 CEGUI::WindowProperties::Text, CEGUI::WindowProperties::Tooltip, CEGUI::WindowProperties::UnifiedAreaRect,  
 CEGUI::WindowProperties::UnifiedHeight, CEGUI::WindowProperties::UnifiedMaxSize,  
 CEGUI::WindowProperties::UnifiedMinSize, CEGUI::WindowProperties::UnifiedPosition,  
 CEGUI::WindowProperties::UnifiedSize, CEGUI::WindowProperties::UnifiedWidth,  
 CEGUI::WindowProperties::UnifiedXPosition, CEGUI::WindowProperties::UnifiedYPosition,  
 CEGUI::WindowProperties::VerticalAlignment, CEGUI::WindowProperties::Visible,  
 CEGUI::WindowProperties::WantsMultiClickEvents, CEGUI::WindowProperties::WindowRenderer,  
 and CEGUI::WindowProperties::ZOrderChangeEnabled.

Collaboration diagram for CEGUI::Property:



## Public Member Functions

- **Property** (const **String** &name, const **String** &help, const **String** &defaultValue="", bool writesXML=true)  
*Creates a new **Property** object.*
- virtual **~Property** (void)  
*Destructor for **Property** objects.*
- const **String** & **getHelp** (void) const  
*Return a **String** that describes the purpose and usage of this **Property**.*
- const **String** & **getName** (void) const  
*Return a the name of this **Property**.*
- virtual **String** **get** (const **PropertyReceiver** \*receiver) const =0  
*Return the current value of the **Property** as a **String**.*
- virtual void **set** (**PropertyReceiver** \*receiver, const **String** &value)=0  
*Sets the value of the property.*
- virtual bool **isDefault** (const **PropertyReceiver** \*receiver) const  
*Returns whether the property is at it's default value.*
- virtual **String** **getDefault** (const **PropertyReceiver** \*receiver) const  
*Returns the default value of the **Property** as a **String**.*
- virtual void **writeXMLToStream** (const **PropertyReceiver** \*receiver, **XMLSerializer** &xml\_stream) const  
*Writes out an XML representation of this class to the given stream.*

## Protected Attributes

- **String** d\_name  
***String** that stores the **Property** name.*
- **String** d\_help  
***String** that stores the **Property** help text.*
- **String** d\_default  
***String** that stores the **Property** default value string.*
- bool d\_writeXML  
*Specifies whether writeXMLToStream should do anything for this property.*

### 6.202.1 Detailed Description

An abstract class that defines the interface to access object properties by name.

[Property](#) objects allow (via a [PropertySet](#)) access to certain properties of objects by using simple get/set functions and the name of the property to be accessed.

### 6.202.2 Constructor & Destructor Documentation

#### 6.202.2.1 CEGUI::Property::Property (const String & *name*, const String & *help*, const String & *defaultValue* = "", bool *writesXML* = true) [inline]

Creates a new [Property](#) object.

##### Parameters:

*name* [String](#) containing the name of the new [Property](#).

*help* [String](#) containing a description of the [Property](#) and it's usage.

*defaultValue* [String](#) holding the textual representation of the default value for this [Property](#)

*writesXML* Specifies whether the writeXMLToStream method should do anything for this [Property](#).  
This enables selectivity in what properties within a [PropertySet](#) will get output as XML.

### 6.202.3 Member Function Documentation

#### 6.202.3.1 const String& CEGUI::Property::getHelp (void) const [inline]

Return a [String](#) that describes the purpose and usage of this [Property](#).

##### Returns:

[String](#) that contains the help text

#### 6.202.3.2 const String& CEGUI::Property::getName (void) const [inline]

Return a the name of this [Property](#).

##### Returns:

[String](#) containing the name of the [Property](#)

#### 6.202.3.3 virtual String CEGUI::Property::get (const PropertyReceiver \* *receiver*) const [pure virtual]

Return the current value of the [Property](#) as a [String](#).

##### Parameters:

*receiver* Pointer to the target object.



**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implemented in CEGUI::WindowProperties::ID, CEGUI::WindowProperties::Alpha, CEGUI::WindowProperties::Font, CEGUI::WindowProperties::Text, CEGUI::WindowProperties::MouseCursorImage, CEGUI::WindowProperties::ClippedByParent, CEGUI::WindowProperties::InheritsAlpha, CEGUI::WindowProperties::AlwaysOnTop, CEGUI::WindowProperties::Disabled, CEGUI::WindowProperties::Visible, CEGUI::WindowProperties::RestoreOldCapture, CEGUI::WindowProperties::DestroyedByParent, CEGUI::WindowProperties::ZOrderChangeEnabled, CEGUI::WindowProperties::WantsMultiClickEvents, CEGUI::WindowProperties::MouseButtonDownAutoRepeat, CEGUI::WindowProperties::AutoRepeatDelay, CEGUI::WindowProperties::AutoRepeatRate, CEGUI::WindowProperties::DistributeCapturedInputs, CEGUI::WindowProperties::CustomTooltipType, CEGUI::WindowProperties::Tooltip, CEGUI::WindowProperties::InheritsTooltipText, CEGUI::WindowProperties::RiseOnClick, CEGUI::WindowProperties::VerticalAlignment, CEGUI::WindowProperties::HorizontalAlignment, CEGUI::WindowProperties::UnifiedAreaRect, CEGUI::WindowProperties::UnifiedPosition, CEGUI::WindowProperties::UnifiedXPosition, CEGUI::WindowProperties::UnifiedYPosition, CEGUI::WindowProperties::UnifiedSize, CEGUI::WindowProperties::UnifiedWidth, CEGUI::WindowProperties::UnifiedHeight, CEGUI::WindowProperties::UnifiedMinSize, CEGUI::WindowProperties::UnifiedMaxSize, CEGUI::WindowProperties::MousePassThroughEnabled, CEGUI::WindowProperties::WindowRenderer, CEGUI::WindowProperties::LookNFeel, CEGUI::WindowProperties::DragDropTarget, CEGUI::CheckboxProperties::Selected, CEGUI::ComboboxProperties::ReadOnly, CEGUI::ComboboxProperties::ValidationString, CEGUI::ComboboxProperties::CaratIndex, CEGUI::ComboboxProperties::EditSelectionStart, CEGUI::ComboboxProperties::EditSelectionLength, CEGUI::ComboboxProperties::MaxEditTextLength, CEGUI::ComboboxProperties::SortList, CEGUI::ComboboxProperties::ForceVertScrollbar, CEGUI::ComboboxProperties::ForceHorzScrollbar, CEGUI::ComboboxProperties::SingleClickMode, CEGUI::DragContainerProperties::DraggingEnabled, CEGUI::DragContainerProperties::DragAlpha, CEGUI::DragContainerProperties::DragThreshold, CEGUI::DragContainerProperties::DragCursorImage, CEGUI::EditboxProperties::ReadOnly, CEGUI::EditboxProperties::MaskText, CEGUI::EditboxProperties::MaskCodepoint, CEGUI::EditboxProperties::ValidationString, CEGUI::EditboxProperties::CaratIndex, CEGUI::EditboxProperties::SelectionStart, CEGUI::EditboxProperties::SelectionLength, CEGUI::EditboxProperties::MaxTextLength, CEGUI::EditboxProperties::NormalTextColour, CEGUI::EditboxProperties::SelectedTextColour, CEGUI::EditboxProperties::ActiveSelectionColour, CEGUI::EditboxProperties::InactiveSelectionColour, CEGUI::FrameWindowProperties::SizingEnabled, CEGUI::FrameWindowProperties::FrameEnabled, CEGUI::FrameWindowProperties::TitlebarEnabled, CEGUI::FrameWindowProperties::CloseButtonEnabled, CEGUI::FrameWindowProperties::RollUpEnabled, CEGUI::FrameWindowProperties::RollUpState, CEGUI::FrameWindowProperties::DragMovingEnabled, CEGUI::FrameWindowProperties::SizingBorderThickness, CEGUI::FrameWindowProperties::NSSizingCursorImage, CEGUI::FrameWindowProperties::EWSizingCursorImage, CEGUI::FrameWindowProperties::NWSEizingCursorImage, CEGUI::FrameWindowProperties::NESWSizingCursorImage, CEGUI::ItemEntryProperties::Selectable, CEGUI::ItemEntryProperties::Selected, CEGUI::ItemListBaseProperties::AutoResizeEnabled, CEGUI::ItemListBaseProperties::SortEnabled, CEGUI::ItemListBaseProperties::SortMode, CEGUI::ItemListboxProperties::MultiSelect, CEGUI::ListboxProperties::Sort, CEGUI::ListboxProperties::MultiSelect, CEGUI::ListboxProperties::ForceVertScrollbar, CEGUI::ListboxProperties::ForceHorzScrollbar, CEGUI::ListboxProperties::ItemTooltips, CEGUI::ListHeaderProperties::ColumnsSizable, CEGUI::ListHeaderProperties::ColumnsMovable, CEGUI::ListHeaderProperties::SortSettingEnabled, CEGUI::ListHeaderProperties::SortDirection, CEGUI::ListHeaderProperties::SortColumnID, CEGUI::ListHeaderSegmentProperties::Sizable, CEGUI::ListHeaderSegmentProperties::Clickable, CEGUI::ListHeaderSegmentProperties::Dragable, CEGUI::ListHeaderSegmentProperties::SortDirection, CEGUI::ListHeaderSegmentProperties::SizingCursorImage, CEGUI::ListHeaderSegmentProperties::MovingCursorImage, CEGUI::MenuBaseProperties::ItemSpacing, CEGUI::MenuBaseProperties::AllowMultiplePopups, CEGUI::MultiColumnListProperties::ColumnsSizable,

CEGUI::MultiColumnListProperties::ColumnsMovable, CEGUI::MultiColumnListProperties::SortSettingEnabled, CEGUI::MultiColumnListProperties::SortDirection, CEGUI::MultiColumnListProperties::SortColumnID, CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, CEGUI::MultiColumnListProperties::NominatedSelection, CEGUI::MultiColumnListProperties::ForceVertScrollbar, CEGUI::MultiColumnListProperties::ForceHorzScrollbar, CEGUI::MultiColumnListProperties::SelectionMode, CEGUI::MultiColumnListProperties::ColumnHeader, CEGUI::MultiColumnListProperties::RowCount, CEGUI::MultiLineEditboxProperties::ReadOnly, CEGUI::MultiLineEditboxProperties::WordWrap, CEGUI::MultiLineEditboxProperties::CaratIndex, CEGUI::MultiLineEditboxProperties::SelectionStart, CEGUI::MultiLineEditboxProperties::SelectionLength, CEGUI::MultiLineEditboxProperties::MaxTextLength, CEGUI::MultiLineEditboxProperties::SelectionBrushImage, CEGUI::MultiLineEditboxProperties::ForceVertScrollbar, CEGUI::PopupMenuProperties::FadeInTime, CEGUI::PopupMenuProperties::FadeOutTime, CEGUI::ProgressBarProperties::CurrentProgress, CEGUI::ProgressBarProperties::StepSize, CEGUI::RadioButtonProperties::Selected, CEGUI::RadioButtonProperties::GroupID, CEGUI::ScrollablePaneProperties::ContentPaneAutoSized, CEGUI::ScrollablePaneProperties::ContentArea, CEGUI::ScrollablePaneProperties::ForceVertScrollbar, CEGUI::ScrollablePaneProperties::ForceHorzScrollbar, CEGUI::ScrollablePaneProperties::HorzStepSize, CEGUI::ScrollablePaneProperties::HorzOverlapSize, CEGUI::ScrollablePaneProperties::HorzScrollPosition, CEGUI::ScrollablePaneProperties::VertStepSize, CEGUI::ScrollablePaneProperties::VertOverlapSize, CEGUI::ScrollablePaneProperties::VertScrollPosition, CEGUI::ScrollbarProperties::DocumentSize, CEGUI::ScrollbarProperties::PageSize, CEGUI::ScrollbarProperties::StepSize, CEGUI::ScrollbarProperties::OverlapSize, CEGUI::ScrollbarProperties::ScrollPosition, CEGUI::ScrolledContainerProperties::ContentPaneAutoSized, CEGUI::ScrolledContainerProperties::ContentArea, CEGUI::ScrolledContainerProperties::ChildExtentsArea, CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar, CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar, CEGUI::SliderProperties::CurrentValue, CEGUI::SliderProperties::MaximumValue, CEGUI::SliderProperties::ClickStepSize, CEGUI::SpinnerProperties::CurrentValue, CEGUI::SpinnerProperties::StepSize, CEGUI::SpinnerProperties::MinimumValue, CEGUI::SpinnerProperties::MaximumValue, CEGUI::SpinnerProperties::TextInputMode, CEGUI::TabControlProperties::TabHeight, CEGUI::TabControlProperties::TabTextPadding, CEGUI::TabControlProperties::TabPagePosition, CEGUI::ThumbProperties::HotTracked, CEGUI::ThumbProperties::VertFree, CEGUI::ThumbProperties::HorzFree, CEGUI::ThumbProperties::VertRange, CEGUI::ThumbProperties::HorzRange, CEGUI::TitlebarProperties::DraggingEnabled, CEGUI::TooltipProperties::HoverTime, CEGUI::TooltipProperties::DisplayTime, CEGUI::TooltipProperties::FadeTime, CEGUI::TreeProperties::Sort, CEGUI::TreeProperties::MultiSelect, CEGUI::TreeProperties::ForceVertScrollbar, CEGUI::TreeProperties::ForceHorzScrollbar, CEGUI::TreeProperties::ItemTooltips, CEGUI::PropertyDefinition, and CEGUI::PropertyLinkDefinition.

#### 6.202.3.4 virtual void CEGUI::Property::set (PropertyReceiver \* *receiver*, const String & *value*) [pure virtual]

Sets the value of the property.

##### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

##### Returns:

Nothing.

##### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implemented in CEGUI::WindowProperties::ID, CEGUI::WindowProperties::Alpha, CEGUI::WindowProperties::Font, CEGUI::WindowProperties::Text, CEGUI::WindowProperties::MouseCursorImage, CEGUI::WindowProperties::ClippedByParent, CEGUI::WindowProperties::InheritsAlpha, CEGUI::WindowProperties::AlwaysOnTop, CEGUI::WindowProperties::Disabled, CEGUI::WindowProperties::Visible, CEGUI::WindowProperties::RestoreOldCapture, CEGUI::WindowProperties::DestroyedByParent, CEGUI::WindowProperties::ZOrderChangeEnabled, CEGUI::WindowProperties::WantsMultiClickEvents, CEGUI::WindowProperties::MouseButtonDownAutoRepeat, CEGUI::WindowProperties::AutoRepeatDelay, CEGUI::WindowProperties::AutoRepeatRate, CEGUI::WindowProperties::DistributeCapturedInputs, CEGUI::WindowProperties::CustomTooltipType, CEGUI::WindowProperties::Tooltip, CEGUI::WindowProperties::InheritsTooltipText, CEGUI::WindowProperties::RiseOnClick, CEGUI::WindowProperties::VerticalAlignment, CEGUI::WindowProperties::HorizontalAlignment, CEGUI::WindowProperties::UnifiedAreaRect, CEGUI::WindowProperties::UnifiedPosition, CEGUI::WindowProperties::UnifiedXPosition, CEGUI::WindowProperties::UnifiedYPosition, CEGUI::WindowProperties::UnifiedSize, CEGUI::WindowProperties::UnifiedWidth, CEGUI::WindowProperties::UnifiedHeight, CEGUI::WindowProperties::UnifiedMinSize, CEGUI::WindowProperties::UnifiedMaxSize, CEGUI::WindowProperties::MousePassThroughEnabled, CEGUI::WindowProperties::WindowRenderer, CEGUI::WindowProperties::LookNFeel, CEGUI::WindowProperties::DragDropTarget, CEGUI::CheckboxProperties::Selected, CEGUI::ComboboxProperties::ReadOnly, CEGUI::ComboboxProperties::ValidationString, CEGUI::ComboboxProperties::CaratIndex, CEGUI::ComboboxProperties::EditSelectionStart, CEGUI::ComboboxProperties::EditSelectionLength, CEGUI::ComboboxProperties::MaxEditTextLength, CEGUI::ComboboxProperties::SortList, CEGUI::ComboboxProperties::ForceVertScrollbar, CEGUI::ComboboxProperties::ForceHorzScrollbar, CEGUI::ComboboxProperties::SingleClickMode, CEGUI::DragContainerProperties::DraggingEnabled, CEGUI::DragContainerProperties::DragAlpha, CEGUI::DragContainerProperties::DragThreshold, CEGUI::DragContainerProperties::DragCursorImage, CEGUI::EditboxProperties::ReadOnly, CEGUI::EditboxProperties::MaskText, CEGUI::EditboxProperties::MaskCodepoint, CEGUI::EditboxProperties::ValidationString, CEGUI::EditboxProperties::CaratIndex, CEGUI::EditboxProperties::SelectionStart, CEGUI::EditboxProperties::SelectionLength, CEGUI::EditboxProperties::MaxTextLength, CEGUI::EditboxProperties::NormalTextColour, CEGUI::EditboxProperties::SelectedTextColour, CEGUI::EditboxProperties::ActiveSelectionColour, CEGUI::EditboxProperties::InactiveSelectionColour, CEGUI::FrameWindowProperties::SizingEnabled, CEGUI::FrameWindowProperties::FrameEnabled, CEGUI::FrameWindowProperties::TitlebarEnabled, CEGUI::FrameWindowProperties::CloseButtonEnabled, CEGUI::FrameWindowProperties::RollUpEnabled, CEGUI::FrameWindowProperties::RollUpState, CEGUI::FrameWindowProperties::DragMovingEnabled, CEGUI::FrameWindowProperties::SizingBorderThickness, CEGUI::FrameWindowProperties::NSSizingCursorImage, CEGUI::FrameWindowProperties::EWSizingCursorImage, CEGUI::FrameWindowProperties::NWSEizingCursorImage, CEGUI::FrameWindowProperties::NESWSizingCursorImage, CEGUI::ItemEntryProperties::Selectable, CEGUI::ItemEntryProperties::Selected, CEGUI::ItemListBaseProperties::AutoResizeEnabled, CEGUI::ItemListBaseProperties::SortEnabled, CEGUI::ItemListBaseProperties::SortMode, CEGUI::ListboxProperties::MultiSelect, CEGUI::ListboxProperties::Sort, CEGUI::ListboxProperties::MultiSelect, CEGUI::ListboxProperties::ForceVertScrollbar, CEGUI::ListboxProperties::ForceHorzScrollbar, CEGUI::ListboxProperties::ItemTooltips, CEGUI::ListHeaderProperties::ColumnsSizable, CEGUI::ListHeaderProperties::ColumnsMovable, CEGUI::ListHeaderProperties::SortSettingEnabled, CEGUI::ListHeaderProperties::SortDirection, CEGUI::ListHeaderProperties::SortColumnID, CEGUI::ListHeaderSegmentProperties::Sizable, CEGUI::ListHeaderSegmentProperties::Clickable, CEGUI::ListHeaderSegmentProperties::Dragable, CEGUI::ListHeaderSegmentProperties::SortDirection, CEGUI::ListHeaderSegmentProperties::SizingCursorImage, CEGUI::ListHeaderSegmentProperties::MovingCursorImage, CEGUI::MenuBaseProperties::ItemSpacing, CEGUI::MenuBaseProperties::AllowMultiplePopups, CEGUI::MultiColumnListProperties::ColumnsSizable, CEGUI::MultiColumnListProperties::ColumnsMovable, CEGUI::MultiColumnListProperties::SortSettingEnabled, CEGUI::MultiColumnListProperties::SortDirection, CEGUI::MultiColumnListProperties::SortColumnID, CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, CEGUI::MultiColumnListProperties::ForceVertScrollbar, CEGUI::MultiColumnListProperties::ForceHorzScrollbar,

CEGUI::MultiColumnListProperties::SelectionMode, CEGUI::MultiColumnListProperties::ColumnHeader,  
 CEGUI::MultiColumnListProperties::RowCount, CEGUI::MultiLineEditboxProperties::ReadOnly,  
 CEGUI::MultiLineEditboxProperties::WordWrap, CEGUI::MultiLineEditboxProperties::CaratIndex,  
 CEGUI::MultiLineEditboxProperties::SelectionStart, CEGUI::MultiLineEditboxProperties::SelectionLength,  
 CEGUI::MultiLineEditboxProperties::MaxTextLength, CEGUI::MultiLineEditboxProperties::SelectionBrushImage,  
 CEGUI::MultiLineEditboxProperties::ForceVertScrollbar, CEGUI::PopupMenuProperties::FadeInTime,  
 CEGUI::PopupMenuProperties::FadeOutTime, CEGUI::ProgressBarProperties::CurrentProgress,  
 CEGUI::ProgressBarProperties::StepSize, CEGUI::RadioButtonProperties::Selected,  
 CEGUI::RadioButtonProperties::GroupID, CEGUI::ScrollablePaneProperties::ContentPaneAutoSized,  
 CEGUI::ScrollablePaneProperties::ContentArea, CEGUI::ScrollablePaneProperties::ForceVertScrollbar,  
 CEGUI::ScrollablePaneProperties::ForceHorzScrollbar, CEGUI::ScrollablePaneProperties::HorzStepSize,  
 CEGUI::ScrollablePaneProperties::HorzOverlapSize, CEGUI::ScrollablePaneProperties::HorzScrollPosition,  
 CEGUI::ScrollablePaneProperties::VertStepSize, CEGUI::ScrollablePaneProperties::VertOverlapSize,  
 CEGUI::ScrollablePaneProperties::VertScrollPosition, CEGUI::ScrollbarProperties::DocumentSize,  
 CEGUI::ScrollbarProperties::PageSize, CEGUI::ScrollbarProperties::StepSize,  
 CEGUI::ScrollbarProperties::OverlapSize, CEGUI::ScrollbarProperties::ScrollPosition,  
 CEGUI::ScrolledContainerProperties::ContentPaneAutoSized, CEGUI::ScrolledContainerProperties::ContentArea,  
 CEGUI::ScrolledContainerProperties::ChildExtentsArea, CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar,  
 CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar, CEGUI::SliderProperties::CurrentValue,  
 CEGUI::SliderProperties::MaximumValue, CEGUI::SliderProperties::ClickStepSize,  
 CEGUI::SpinnerProperties::CurrentValue, CEGUI::SpinnerProperties::StepSize,  
 CEGUI::SpinnerProperties::MinimumValue, CEGUI::SpinnerProperties::MaximumValue,  
 CEGUI::SpinnerProperties::TextInputMode, CEGUI::TabControlProperties::TabHeight,  
 CEGUI::TabControlProperties::TabTextPadding, CEGUI::TabControlProperties::TabPagePosition,  
 CEGUI::ThumbProperties::HotTracked, CEGUI::ThumbProperties::VertFree,  
 CEGUI::ThumbProperties::HorzFree, CEGUI::ThumbProperties::VertRange,  
 CEGUI::ThumbProperties::HorzRange, CEGUI::TitlebarProperties::DraggingEnabled,  
 CEGUI::TooltipProperties::HoverTime, CEGUI::TooltipProperties::DisplayTime,  
 CEGUI::TooltipProperties::FadeTime, CEGUI::TreeProperties::Sort, CEGUI::TreeProperties::MultiSelect,  
 CEGUI::TreeProperties::ForceVertScrollbar, CEGUI::TreeProperties::ForceHorzScrollbar,  
 CEGUI::TreeProperties::ItemTooltips, CEGUI::PropertyDefinition, CEGUI::PropertyDefinitionBase,  
 and CEGUI::PropertyLinkDefinition.

#### 6.202.3.5 **bool CEGUI::Property::isDefault (const PropertyReceiver \* *receiver*) const** [virtual]

Returns whether the property is at it's default value.

##### Parameters:

***receiver*** Pointer to the target object.

##### Returns:

- true if the property has it's default value.
- false if the property has been modified from it's default value.

Reimplemented in CEGUI::WindowProperties::Font, CEGUI::WindowProperties::MouseCursorImage, CEGUI::WindowProperties::Disabled, and CEGUI::WindowProperties::Visible.

#### 6.202.3.6 **String CEGUI::Property::getDefault (const PropertyReceiver \* *receiver*) const** [virtual]

Returns the default value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the default value for this property.

**6.202.3.7** `void CEGUI::Property::writeXMLToStream (const PropertyReceiver * receiver,  
XMLSerializer & xml_stream) const` [virtual]

Writes out an XML representation of this class to the given stream.

**Note:**

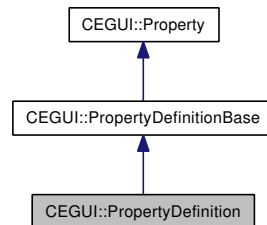
This would normally have been implemented via XMLGenerator base class, but in this case we require the target [PropertyReceiver](#) in order to obtain the property value.

Reimplemented in [CEGUI::WindowProperties::WindowRenderer](#), and [CEGUI::WindowProperties::LookNFeel](#).

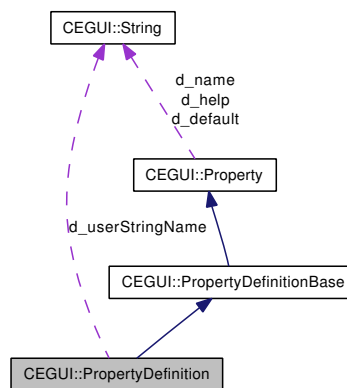
## 6.203 CEGUI::PropertyDefinition Class Reference

Class representing a generic get/set property.

Inheritance diagram for CEGUI::PropertyDefinition:



Collaboration diagram for CEGUI::PropertyDefinition:



### Public Member Functions

- **PropertyDefinition** (const [String](#) &name, const [String](#) &initialValue, bool redrawOnWrite, bool layoutOnWrite)
- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### Protected Member Functions

- void **writeXMLElementType** ([XMLSerializer](#) &xml\_stream) const  
*Write out the text of the XML element type. Note that you should not write the opening '<' character, nor any other information such as attributes in this function. The writeExtraAttributes function can be used for writing attributes.*

## Protected Attributes

- [String](#) `d_userStringName`

### 6.203.1 Detailed Description

Class representing a generic get/set property.

### 6.203.2 Member Function Documentation

#### 6.203.2.1 `String CEGUI::PropertyDefinition::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

##### Parameters:

*receiver* Pointer to the target object.

##### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

#### 6.203.2.2 `void CEGUI::PropertyDefinition::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

##### Note:

When overriding the `set()` member of [PropertyDefinitionBase](#), you MUST call the base class implementation after you have set the property value (i.e. you must call [PropertyDefinitionBase::set\(\)](#)).

##### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

##### Returns:

Nothing.

##### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Reimplemented from [CEGUI::PropertyDefinitionBase](#).

### 6.203.2.3 void CEGUI::PropertyDefinition::writeXMLElementType (XMLSerializer & *xml\_stream*) const [protected, virtual]

Write out the text of the XML element type. Note that you should not write the opening '<' character, nor any other information such as attributes in this function. The writeExtraAttributes function can be used for writing attributes.

#### Parameters:

*xml\_stream* Stream where xml data should be output.

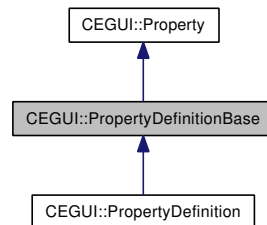
Implements [CEGUI::PropertyDefinitionBase](#).



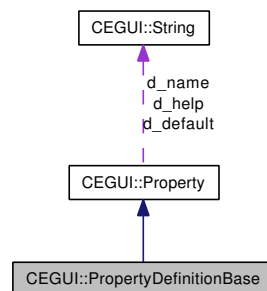
## 6.204 CEGUI::PropertyDefinitionBase Class Reference

common base class used for types representing a new property to be available on all widgets that use the WidgetLook that the property definition is a part of.

Inheritance diagram for CEGUI::PropertyDefinitionBase:



Collaboration diagram for CEGUI::PropertyDefinitionBase:



### Public Member Functions

- **PropertyDefinitionBase** (const [String](#) &name, const [String](#) &help, const [String](#) &initialValue, bool redrawOnWrite, bool layoutOnWrite)
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- virtual void **writeXMLToStream** ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of the [PropertyDefinitionBase](#) based object to out\_stream.*

### Protected Member Functions

- virtual void **writeXMLElementType** ([XMLSerializer](#) &xml\_stream) const =0  
*Write out the text of the XML element type. Note that you should not write the opening '<' character, nor any other information such as attributes in this function. The writeExtraAttributes function can be used for writing attributes.*
- virtual void **writeXMLAttributes** ([XMLSerializer](#) &xml\_stream) const  
*Write out any xml attributes added in a sub-class. Note that you should not write the closing '>' character sequence, nor any other information in this function. You should always call the base class implementation of this function when overriding.*

## Protected Attributes

- bool `d_writeCausesRedraw`
- bool `d_writeCausesLayout`

### 6.204.1 Detailed Description

common base class used for types representing a new property to be available on all widgets that use the WidgetLook that the property definition is a part of.

### 6.204.2 Member Function Documentation

#### 6.204.2.1 `void CEGUI::PropertyDefinitionBase::set (PropertyReceiver * receiver, const String & value) [virtual]`

Sets the value of the property.

#### Note:

When overriding the `set()` member of [PropertyDefinitionBase](#), you MUST call the base class implementation after you have set the property value (i.e. you must call [PropertyDefinitionBase::set\(\)](#)).

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

Reimplemented in [CEGUI::PropertyDefinition](#), and [CEGUI::PropertyLinkDefinition](#).

#### 6.204.2.2 `void CEGUI::PropertyDefinitionBase::writeXMLToStream (XMLSerializer & xml_stream) const [virtual]`

Writes an xml representation of the [PropertyDefinitionBase](#) based object to *out\_stream*.

#### Parameters:

*xml\_stream* Stream where xml data should be output.

#### Returns:

Nothing.

**6.204.2.3 virtual void CEGUI::PropertyDefinitionBase::writeXMLElementType (XMLSerializer & *xml\_stream*) const** [protected, pure virtual]

Write out the text of the XML element type. Note that you should not write the opening '<' character, nor any other information such as attributes in this function. The writeExtraAttributes function can be used for writing attributes.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

Implemented in [CEGUI::PropertyDefinition](#), and [CEGUI::PropertyLinkDefinition](#).

**6.204.2.4 void CEGUI::PropertyDefinitionBase::writeXMLAttributes (XMLSerializer & *xml\_stream*) const** [protected, virtual]

Write out any xml attributes added in a sub-class. Note that you should not write the closing '>' character sequence, nor any other information in this function. You should always call the base class implementation of this function when overriding.

**Parameters:**

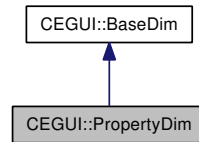
*xml\_stream* Stream where xml data should be output.

Reimplemented in [CEGUI::PropertyLinkDefinition](#).

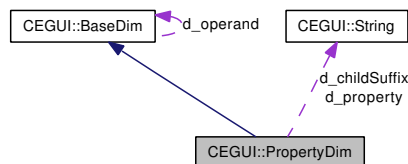
## 6.205 CEGUI::PropertyDim Class Reference

**Dimension** type that represents the value of a **Window** property. Implements **BaseDim** interface.

Inheritance diagram for CEGUI::PropertyDim:



Collaboration diagram for CEGUI::PropertyDim:



### Public Member Functions

- **PropertyDim** (const **String** &name, const **String** &property, **DimensionType** type)

*Constructor.*

### Protected Member Functions

- float **getValue\_impl** (const **Window** &wnd) const

*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically.*

- float **getValue\_impl** (const **Window** &wnd, const **Rect** &container) const

*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically by **BaseDim**.*

- void **writeXMLElementName\_impl** (**XMLSerializer** &xml\_stream) const

*Implementataion method to output real xml element name.*

- void **writeXMLElementAttributes\_impl** (**XMLSerializer** &xml\_stream) const

*Implementataion method to create the element attributes.*

- **BaseDim** \* **clone\_impl** () const

*Implementataion method to return a clone of this sub-class of **BaseDim**. This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by **BaseDim**.*

## 6.205.1 Detailed Description

[Dimension](#) type that represents the value of a [Window](#) property. Implements [BaseDim](#) interface.

## 6.205.2 Constructor & Destructor Documentation

### 6.205.2.1 CEGUI::PropertyDim::PropertyDim (const String & *name*, const String & *property*, DimensionType *type*)

Constructor.

#### Parameters:

*name* [String](#) holding the name suffix of the window on which the property is to be accessed.

*property* [String](#) object holding the name of the property this [PropertyDim](#) represents the value of. The property named should represent a simple float value.

*type* DimensionType value indicating what dimension named property represents.

## 6.206 CEGUI::PropertyHelper Class Reference

Helper class used to convert various data types to and from the format expected in Property strings.

### Static Public Member Functions

- static float **stringToFloat** (const [String](#) &str)
- static uint **stringToUint** (const [String](#) &str)
- static int **stringToInt** (const [String](#) &str)
- static bool **stringToBool** (const [String](#) &str)
- static [Size](#) **stringToSize** (const [String](#) &str)
- static [Point](#) **stringToPoint** (const [String](#) &str)
- static [Rect](#) **stringToRect** (const [String](#) &str)
- static const [Image](#) \* **stringToImage** (const [String](#) &str)
- static [colour](#) **stringToColour** (const [String](#) &str)
- static [ColourRect](#) **stringToColourRect** (const [String](#) &str)
- static [UDim](#) **stringToUDim** (const [String](#) &str)
- static [UVector2](#) **stringToUVector2** (const [String](#) &str)
- static [URect](#) **stringToURect** (const [String](#) &str)
- static [String](#) **floatToString** (float val)
- static [String](#) **uintToString** (uint val)
- static [String](#) **intToString** (int val)
- static [String](#) **boolToString** (bool val)
- static [String](#) **sizeToString** (const [Size](#) &val)
- static [String](#) **pointToString** (const [Point](#) &val)
- static [String](#) **rectToString** (const [Rect](#) &val)
- static [String](#) **imageToString** (const [Image](#) \*const val)
- static [String](#) **colourToString** (const [colour](#) &val)
- static [String](#) **colourRectToString** (const [ColourRect](#) &val)
- static [String](#) **udimToString** (const [UDim](#) &val)
- static [String](#) **uvector2ToString** (const [UVector2](#) &val)
- static [String](#) **urectToString** (const [URect](#) &val)

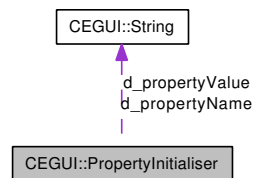
### 6.206.1 Detailed Description

Helper class used to convert various data types to and from the format expected in Property strings.

## 6.207 CEGUI::PropertyInitialiser Class Reference

Class that holds information about a property and it's required initial value.

Collaboration diagram for CEGUI::PropertyInitialiser:



### Public Member Functions

- **PropertyInitialiser** (const [String](#) &property, const [String](#) &value)  
*Constructor.*
- void **apply** ([PropertySet](#) &target) const  
*Apply this property initialiser to the specified target [CEGUI::PropertySet](#) object.*
- const [String](#) & **getTargetPropertyName** () const  
*Return the name of the property targetted by this [PropertyInitialiser](#).*
- const [String](#) & **getInitialiserValue** () const  
*Return the value string to be set on the property targetted by this [PropertyInitialiser](#).*
- void **writeXMLToStream** ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [PropertyInitialiser](#) to out\_stream.*

### 6.207.1 Detailed Description

Class that holds information about a property and it's required initial value.

### 6.207.2 Constructor & Destructor Documentation

#### 6.207.2.1 CEGUI::PropertyInitialiser::PropertyInitialiser (const [String](#) & *property*, const [String](#) & *value*)

Constructor.

#### Parameters:

*property* [String](#) holding the name of the property targetted by this [PropertyInitialiser](#).

*value* [String](#) holding the value to be set by this [PropertyInitialiser](#).

### 6.207.3 Member Function Documentation

#### 6.207.3.1 void CEGUI::PropertyInitialiser::apply (CEGUI::PropertySet & *target*) const

Apply this property initialiser to the specified target [CEGUI::PropertySet](#) object.

**Parameters:**

*target* [CEGUI::PropertySet](#) object to be initialised by this [PropertyInitialiser](#).

**Returns:**

Nothing.

#### 6.207.3.2 const String & CEGUI::PropertyInitialiser::getTargetPropertyName () const

Return the name of the property targetted by this [PropertyInitialiser](#).

**Returns:**

[String](#) object holding the name of the target property.

#### 6.207.3.3 const String & CEGUI::PropertyInitialiser::getInitialiserValue () const

Return the value string to be set on the property targetted by this [PropertyInitialiser](#).

**Returns:**

[String](#) object holding the value string.

#### 6.207.3.4 void CEGUI::PropertyInitialiser::writeXMLToStream (XMLSerializer & *xml\_stream*) const

Writes an xml representation of this [PropertyInitialiser](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

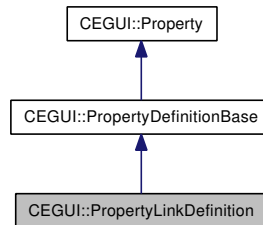
Nothing.



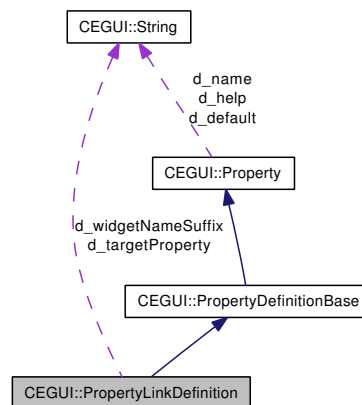
## 6.208 CEGUI::PropertyLinkDefinition Class Reference

Class representing a property that links to another property defined on an attached child widget.

Inheritance diagram for CEGUI::PropertyLinkDefinition:



Collaboration diagram for CEGUI::PropertyLinkDefinition:



### Public Member Functions

- **PropertyLinkDefinition** (const [String](#) &propertyName, const [String](#) &widgetNameSuffix, const [String](#) &targetProperty, const [String](#) &initialValue, bool redrawOnWrite, bool layoutOnWrite)
- **String** **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*

- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### Protected Member Functions

- void **writeXMLElementType** ([XMLSerializer](#) &xml\_stream) const  
*Write out the text of the XML element type. Note that you should not write the opening '<' character, nor any other information such as attributes in this function. The writeExtraAttributes function can be used for writing attributes.*
- void **writeXMLAttributes** ([XMLSerializer](#) &xml\_stream) const

*Write out any xml attributes added in a sub-class. Note that you should not write the closing '>' character sequence, nor any other information in this function. You should always call the base class implementation of this function when overriding.*

- `const Window * getTargetWindow (const PropertyReceiver *receiver) const`  
*return a pointer to the window containing the target property to be accessed.*
- `Window * getTargetWindow (PropertyReceiver *receiver)`

## Protected Attributes

- `String d_widgetNameSuffix`
- `String d_targetProperty`

### 6.208.1 Detailed Description

Class representing a property that links to another property defined on an attached child widget.

### 6.208.2 Member Function Documentation

#### 6.208.2.1 `String CEGUI::PropertyLinkDefinition::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the `Property` as a `String`.

##### Parameters:

*receiver* Pointer to the target object.

##### Returns:

`String` object containing a textual representation of the current value of the `Property`

Implements `CEGUI::Property`.

#### 6.208.2.2 `void CEGUI::PropertyLinkDefinition::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

##### Note:

When overriding the `set()` member of `PropertyDefinitionBase`, you MUST call the base class implementation after you have set the property value (i.e. you must call `PropertyDefinitionBase::set()`).

##### Parameters:

*receiver* Pointer to the target object.

*value* A `String` object that contains a textual representation of the new value to assign to the `Property`.

##### Returns:

Nothing.

**Exceptions:**

*InvalidRequestException* Thrown when the [Property](#) was unable to interpret the content of *value*.

Reimplemented from [CEGUI::PropertyDefinitionBase](#).

**6.208.2.3 void CEGUI::PropertyLinkDefinition::writeXMLElementType (XMLSerializer & *xml\_stream*) const** [protected, virtual]

Write out the text of the XML element type. Note that you should not write the opening '<' character, nor any other information such as attributes in this function. The writeExtraAttributes function can be used for writing attributes.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

Implements [CEGUI::PropertyDefinitionBase](#).

**6.208.2.4 void CEGUI::PropertyLinkDefinition::writeXMLAttributes (XMLSerializer & *xml\_stream*) const** [protected, virtual]

Write out any xml attributes added in a sub-class. Note that you should not write the closing '>' character sequence, nor any other information in this function. You should always call the base class implementation of this function when overriding.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

Reimplemented from [CEGUI::PropertyDefinitionBase](#).

**6.208.2.5 const Window \* CEGUI::PropertyLinkDefinition::getTargetWindow (const PropertyReceiver \* *receiver*) const** [protected]

return a pointer to the window containing the target property to be accessed.

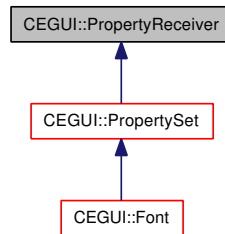
**Exceptions:**

*UnknownObjectException* thrown if no such target window exists within the system.

## 6.209 CEGUI::PropertyReceiver Class Reference

Dummy base class to ensure correct casting of receivers.

Inheritance diagram for CEGUI::PropertyReceiver:



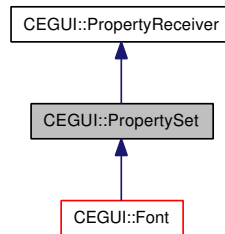
### 6.209.1 Detailed Description

Dummy base class to ensure correct casting of receivers.

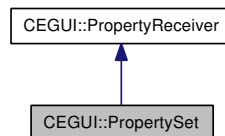
## 6.210 CEGUI::PropertySet Class Reference

Class that contains a collection of [Property](#) objects.

Inheritance diagram for CEGUI::PropertySet:



Collaboration diagram for CEGUI::PropertySet:



### Public Types

- typedef [ConstBaseIterator](#)< PropertyRegistry > **Iterator**

### Public Member Functions

- [PropertySet](#) (void)  
*Constructs a new [PropertySet](#) object.*
- virtual [~PropertySet](#) (void)  
*Destructor for [PropertySet](#) objects.*
- void [addProperty](#) ([Property](#) \*property)  
*Adds a new [Property](#) to the [PropertySet](#).*
- void [removeProperty](#) (const [String](#) &name)  
*Removes a [Property](#) from the [PropertySet](#).*
- void [clearProperties](#) (void)  
*Removes all [Property](#) objects from the [PropertySet](#).*
- bool [isPropertyPresent](#) (const [String](#) &name) const  
*Checks to see if a [Property](#) with the given name is in the [PropertySet](#).*
- const [String](#) & [getPropertyHelp](#) (const [String](#) &name) const  
*Return the help text for the specified [Property](#).*

- [String](#) `getProperty` (const [String](#) &name) const  
*Gets the current value of the specified [Property](#).*
- void `setProperty` (const [String](#) &name, const [String](#) &value)  
*Sets the current value of a [Property](#).*
- bool `isPropertyDefault` (const [String](#) &name) const  
*Returns whether a [Property](#) is at it's default value.*
- [String](#) `getPropertyDefault` (const [String](#) &name) const  
*Returns the default value of a [Property](#) as a [String](#).*
- [Iterator](#) `getIterator` (void) const  
*Return a `PropertySet::Iterator` object to iterate over the available [Properties](#).*

## 6.210.1 Detailed Description

Class that contains a collection of [Property](#) objects.

## 6.210.2 Member Function Documentation

### 6.210.2.1 void CEGUI::PropertySet::addProperty (Property \* *property*)

Adds a new [Property](#) to the [PropertySet](#).

#### Parameters:

*property* Pointer to the [Property](#) object to be added to the [PropertySet](#).

#### Returns:

Nothing.

#### Exceptions:

[NullObjectException](#) Thrown if *property* is NULL.

[AlreadyExistsException](#) Thrown if a [Property](#) with the same name as *property* already exists in the [PropertySet](#)

### 6.210.2.2 void CEGUI::PropertySet::removeProperty (const String & *name*)

Removes a [Property](#) from the [PropertySet](#).

#### Parameters:

*name* [String](#) containing the name of the [Property](#) to be removed. If [Property](#) *name* is not in the set, nothing happens.

#### Returns:

Nothing.

**6.210.2.3 void CEGUI::PropertySet::clearProperties (void)**

Removes all [Property](#) objects from the [PropertySet](#).

**Returns:**

Nothing.

**6.210.2.4 bool CEGUI::PropertySet::isPropertyPresent (const String & name) const**

Checks to see if a [Property](#) with the given name is in the [PropertySet](#).

**Parameters:**

*name* [String](#) containing the name of the [Property](#) to check for.

**Returns:**

true if a [Property](#) named *name* is in the [PropertySet](#). false if no [Property](#) named *name* is in the [PropertySet](#).

**6.210.2.5 const String & CEGUI::PropertySet::getPropertyHelp (const String & name) const**

Return the help text for the specified [Property](#).

**Parameters:**

*name* [String](#) holding the name of the [Property](#) who's help text is to be returned.

**Returns:**

[String](#) object containing the help text for the [Property](#) *name*.

**Exceptions:**

[UnknownObjectException](#) Thrown if no [Property](#) named *name* is in the [PropertySet](#).

**6.210.2.6 String CEGUI::PropertySet::getProperty (const String & name) const**

Gets the current value of the specified [Property](#).

**Parameters:**

*name* [String](#) containing the name of the [Property](#) who's value is to be returned.

**Returns:**

[String](#) object containing a textual representation of the requested [Property](#).

**Exceptions:**

[UnknownObjectException](#) Thrown if no [Property](#) named *name* is in the [PropertySet](#).

**6.210.2.7 void CEGUI::PropertySet::setProperty (const String & *name*, const String & *value*)**

Sets the current value of a [Property](#).

**Parameters:**

*name* [String](#) containing the name of the [Property](#) who's value is to be set.

*value* [String](#) containing a textual representation of the new value for the [Property](#)

**Returns:**

Nothing

**Exceptions:**

[UnknownObjectException](#) Thrown if no [Property](#) named *name* is in the [PropertySet](#).

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

**6.210.2.8 bool CEGUI::PropertySet::isPropertyDefault (const String & *name*) const**

Returns whether a [Property](#) is at it's default value.

**Parameters:**

*name* [String](#) containing the name of the [Property](#) who's default state is to be tested.

**Returns:**

- true if the property has it's default value.
- false if the property has been modified from it's default value.

**6.210.2.9 String CEGUI::PropertySet::getPropertyDefault (const String & *name*) const**

Returns the default value of a [Property](#) as a [String](#).

**Parameters:**

*name* [String](#) containing the name of the [Property](#) who's default string is to be returned.

**Returns:**

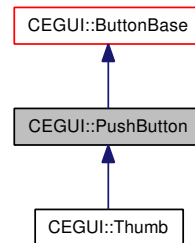
[String](#) object containing a textual representation of the default value for this property.



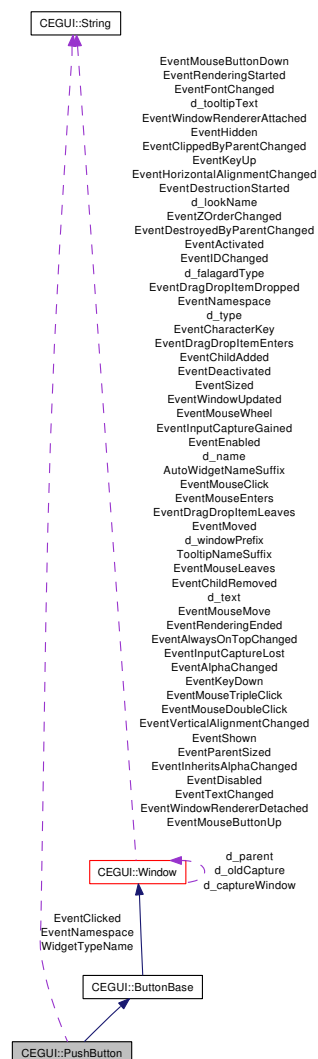
## 6.211 CEGUI::PushButton Class Reference

Base class to provide logic for push button type widgets.

Inheritance diagram for CEGUI::PushButton:



Collaboration diagram for CEGUI::PushButton:



## Public Member Functions

- `PushButton` (const `String` &type, const `String` &name)

*Constructor for base `PushButton` class.*

- virtual `~PushButton` (void)

*Destructor for `PushButton` class.*

## Static Public Attributes

- static const `String EventNamespace`

*Namespace for global events.*

- static const `String WidgetTypeName`

*Window factory name.*

- static const `String EventClicked`

*The button was clicked.*

## Protected Member Functions

- virtual void `onClicked` (`WindowEventArgs` &e)

*handler invoked internally when the button is clicked.*

- virtual void `onMouseButtonUp` (`MouseEventArgs` &e)

*Handler called when a mouse button has been released within this window's area.*

- virtual bool `testClassName_impl` (const `String` &class\_name) const

*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

### 6.211.1 Detailed Description

Base class to provide logic for push button type widgets.

### 6.211.2 Member Function Documentation

#### 6.211.2.1 void CEGUI::PushButton::onMouseButtonUp (MouseEventArgs & e) [protected, virtual]

Handler called when a mouse button has been released within this window's area.

#### Parameters:

*e* `MouseEventArgs` object. All fields are valid.

Reimplemented from `CEGUI::ButtonBase`.

**6.211.2.2** `virtual bool CEGUI::PushButton::testClassName_impl (const String & class_name)`  
`const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

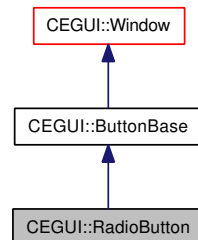
Reimplemented from [CEGUI::ButtonBase](#).

Reimplemented in [CEGUI::Thumb](#).

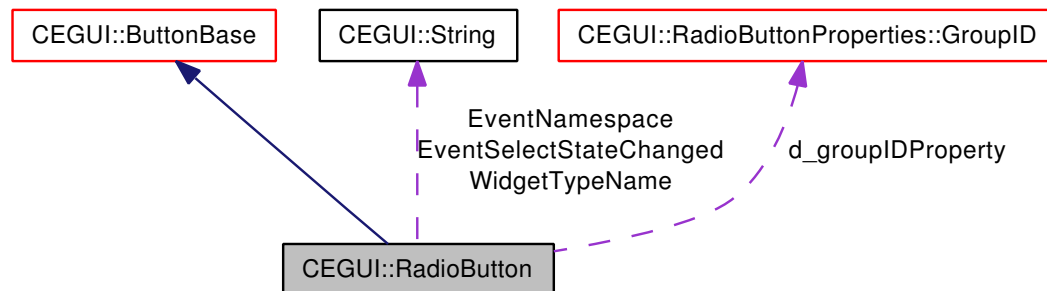
## 6.212 CEGUI::RadioButton Class Reference

Base class to provide the logic for Radio Button widgets.

Inheritance diagram for CEGUI::RadioButton:



Collaboration diagram for CEGUI::RadioButton:



### Public Member Functions

- `bool` [isSelected](#) (void) const  
*return true if the radio button is selected (has the checkmark)*
- `ulong` [getGroupID](#) (void) const  
*return the groupID assigned to this radio button*
- `RadioButton *` [getSelectedButtonInGroup](#) (void) const  
*Return a pointer to the [RadioButton](#) object within the same group as this [RadioButton](#), that is currently selected.*
- `void` [setSelected](#) (bool select)  
*set whether the radio button is selected or not*
- `void` [setGroupID](#) (ulong group)  
*set the groupID for this radio button*
- `RadioButton` (const `String` &type, const `String` &name)

## Static Public Attributes

- static const [String EventNamespace](#)  
*Namespace for global events.*
- static const [String WidgetTypeName](#)  
*Window factory name.*
- static const [String EventSelectStateChanged](#)  
*The selected state of the widget has changed.*

## Protected Member Functions

- void [deselectOtherButtonsInGroup](#) (void) const  
*Deselect any selected radio buttons attached to the same parent within the same group (but not do not deselect 'this').*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [onSelectStateChanged](#) ([WindowEventArgs](#) &e)  
*event triggered internally when the select state of the button changes.*
- virtual void [onMouseButtonUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*

## Protected Attributes

- bool **d\_selected**
- ulong **d\_groupID**

### 6.212.1 Detailed Description

Base class to provide the logic for Radio Button widgets.

### 6.212.2 Member Function Documentation

#### 6.212.2.1 bool CEGUI::RadioButton::isSelected (void) const [inline]

return true if the radio button is selected (has the checkmark)

#### Returns:

true if this widget is selected, false if the widget is not selected.

**6.212.2.2    `ulong CEGUI::RadioButton::getGroupID (void) const`    `[inline]`**

return the groupID assigned to this radio button

**Returns:**

ulong value that identifies the Radio Button group this widget belongs to.

**6.212.2.3    `RadioButton * CEGUI::RadioButton::getSelectedButtonInGroup (void) const`**

Return a pointer to the [RadioButton](#) object within the same group as this [RadioButton](#), that is currently selected.

**Returns:**

Pointer to the [RadioButton](#) object that is the [RadioButton](#) within the same group as this [RadioButton](#), and is attached to the same parent window as this [RadioButton](#), that is currently selected. Returns NULL if no button within the group is selected, or if 'this' is not attached to a parent window.

**6.212.2.4    `void CEGUI::RadioButton::setSelected (bool select)`**

set whether the radio button is selected or not

**Parameters:**

*select* true to put the radio button in the selected state, false to put the radio button in the deselected state. If changing to the selected state, any previously selected radio button within the same group is automatically deselected.

**Returns:**

Nothing.

**6.212.2.5    `void CEGUI::RadioButton::setGroupID (ulong group)`**

set the groupID for this radio button

**Parameters:**

*group* ulong value specifying the radio button group that this widget belongs to.

**Returns:**

Nothing.

**6.212.2.6    `virtual bool CEGUI::RadioButton::testClassName_impl (const String & class_name) const`    `[inline, protected, virtual]`**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::ButtonBase](#).

**6.212.2.7** `void CEGUI::RadioButton::onMouseButtonUp (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

## 6.213 CEGUI::RawDataContainer Class Reference

Class used as the databuffer for loading files throughout the library.

### Public Member Functions

- [RawDataContainer](#) ()  
*Constructor for [RawDataContainer](#) class.*
- [~RawDataContainer](#) (void)  
*Destructor for [RawDataContainer](#) class.*
- void [setData](#) (uint8 \*data)  
*Set a pointer to the external data.*
- uint8 \* [getDataPtr](#) (void)  
*Return a pointer to the external data.*
- const uint8 \* [getDataPtr](#) (void) const
- void [setSize](#) (size\_t size)  
*Set the size of the external data.*
- size\_t [getSize](#) (void) const  
*Get the size of the external data.*
- void [release](#) (void)  
*Release supplied data.*

### 6.213.1 Detailed Description

Class used as the databuffer for loading files throughout the library.

### 6.213.2 Member Function Documentation

#### 6.213.2.1 void CEGUI::RawDataContainer::setData (uint8 \* data) [inline]

Set a pointer to the external data.

##### Parameters:

**data** Pointer to the uint8 data buffer.

#### 6.213.2.2 uint8\* CEGUI::RawDataContainer::getDataPtr (void) [inline]

Return a pointer to the external data.

##### Returns:

Pointer to an the uint8 data buffer.



**6.213.2.3 void CEGUI::RawDataContainer::setSize (size\_t *size*) [inline]**

Set the size of the external data.

**Parameters:**

*size* size\_t containing the size of the external data

**6.213.2.4 size\_t CEGUI::RawDataContainer::getSize (void) const [inline]**

Get the size of the external data.

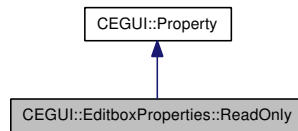
**Returns:**

size\_t containing the size of the external data

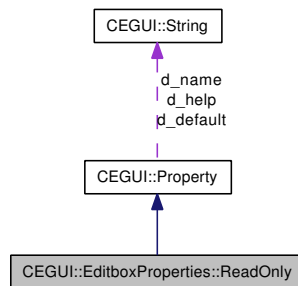
## 6.214 CEGUI::EditboxProperties::ReadOnly Class Reference

[Property](#) to access the read-only setting of the edit box.

Inheritance diagram for CEGUI::EditboxProperties::ReadOnly:



Collaboration diagram for CEGUI::EditboxProperties::ReadOnly:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.214.1 Detailed Description

[Property](#) to access the read-only setting of the edit box.

This property offers access to the read-only setting for the [Editbox](#) object.

#### Usage:

- Name: [ReadOnly](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate the edit box is read-only.
- "False" to indicate the edit box is not read-only (text may be edited by user).

## 6.214.2 Member Function Documentation

### 6.214.2.1 String CEGUI::EditboxProperties::ReadOnly::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.214.2.2 void CEGUI::EditboxProperties::ReadOnly::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

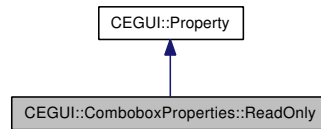
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

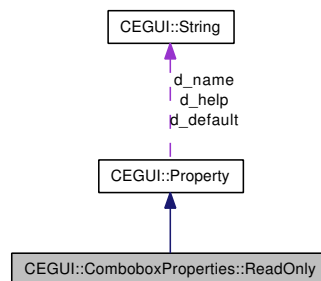
## 6.215 CEGUI::ComboboxProperties::ReadOnly Class Reference

[Property](#) to access the read-only setting of the edit box.

Inheritance diagram for CEGUI::ComboboxProperties::ReadOnly:



Collaboration diagram for CEGUI::ComboboxProperties::ReadOnly:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.215.1 Detailed Description

[Property](#) to access the read-only setting of the edit box.

Usage:

- Name: [ReadOnly](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the edit box is read-only.
- "False" to indicate the edit box is not read-only (text may be edited by user).

## 6.215.2 Member Function Documentation

### 6.215.2.1 String CEGUI::ComboboxProperties::ReadOnly::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.215.2.2 void CEGUI::ComboboxProperties::ReadOnly::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

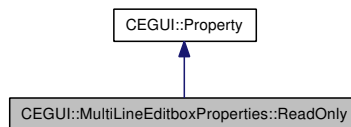
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

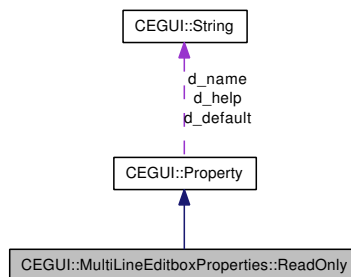
## 6.216 CEGUI::MultiLineEditboxProperties::ReadOnly Class Reference

[Property](#) to access the read-only setting of the edit box.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::ReadOnly:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::ReadOnly:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.216.1 Detailed Description

[Property](#) to access the read-only setting of the edit box.

##### Usage:

- Name: [ReadOnly](#)
- Format: "[text]"

##### Where [Text] is:

- "True" to indicate the edit box is read-only.
- "False" to indicate the edit box is not read-only (text may be edited by user).

## 6.216.2 Member Function Documentation

### 6.216.2.1 String CEGUI::MultiLineEditboxProperties::ReadOnly::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.216.2.2 void CEGUI::MultiLineEditboxProperties::ReadOnly::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.217 CEGUI::Rect Class Reference

Class encapsulating operations on a Rectangle.

### Public Member Functions

- **Rect** (float left, float top, float right, float bottom)  
*Constructor for a [Rect](#).*
- **Rect** ([Point](#) pos, [Size](#) sz)
- [Point](#) **getPosition** (void) const  
*Return top-left position of [Rect](#) as a [Point](#).*
- float **getWidth** (void) const  
*return width of [Rect](#) area*
- float **getHeight** (void) const  
*return height of [Rect](#) area*
- [Size](#) **getSize** (void) const  
*return the size of the [Rect](#) area*
- void **setPosition** (const [Point](#) &pt)  
*set the position of the [Rect](#) (leaves size intact)*
- void **setWidth** (float width)  
*set the width of the [Rect](#) object*
- void **setHeight** (float height)  
*set the height of the [Rect](#) object*
- void **setSize** (const [Size](#) &sz)  
*set the size of the [Rect](#) area*
- [Rect](#) **getIntersection** (const [Rect](#) &rect) const  
*return a [Rect](#) that is the intersection of 'this' [Rect](#) with the [Rect](#) 'rect'*
- [Rect](#) & **offset** (const [Point](#) &pt)  
*Applies an offset to the [Rect](#) object.*
- bool **isPointInRect** (const [Point](#) &pt) const  
*Return true if the given [Point](#) falls within this [Rect](#).*
- [Rect](#) & **constrainSizeMax** (const [Size](#) &sz)  
*check the size of the [Rect](#) object and if it is bigger than sz, resize it so it isn't.*
- [Rect](#) & **constrainSizeMin** (const [Size](#) &sz)  
*check the size of the [Rect](#) object and if it is smaller than sz, resize it so it isn't.*



- [Rect](#) & [constrainSize](#) (const [Size](#) &max\_sz, const [Size](#) &min\_sz)  
*check the size of the [Rect](#) object and if it is bigger than max\_sz or smaller than min\_sz, resize it so it isn't.*
- bool [operator==](#) (const [Rect](#) &rhs) const
- bool [operator!=](#) (const [Rect](#) &rhs) const
- [Rect](#) & [operator=](#) (const [Rect](#) &rhs)
- [Rect](#) [operator](#) \* (float scalar) const
- const [Rect](#) & [operator](#) \*= (float scalar)

## Public Attributes

- float [d\\_top](#)
- float [d\\_bottom](#)
- float [d\\_left](#)
- float [d\\_right](#)

### 6.217.1 Detailed Description

Class encapsulating operations on a Rectangle.

### 6.217.2 Member Function Documentation

#### 6.217.2.1 [Rect](#) CEGUI::Rect::getIntersection (const [Rect](#) & *rect*) const

return a [Rect](#) that is the intersection of 'this' [Rect](#) with the [Rect](#) 'rect'

##### Note:

It can be assumed that if `d_left == d_right`, or `d_top == d_bottom`, or `getWidth() == 0`, or `getHeight() == 0`, then 'this' rect was totally outside 'rect'.

#### 6.217.2.2 [Rect](#) & CEGUI::Rect::offset (const [Point](#) & *pt*)

Applies an offset the [Rect](#) object.

##### Parameters:

*pt* Point object containing the offsets to be applied to the [Rect](#).

##### Returns:

this [Rect](#) after the offset is performed

#### 6.217.2.3 bool CEGUI::Rect::isPointInRect (const [Point](#) & *pt*) const

Return true if the given Point falls within this [Rect](#).

##### Parameters:

*pt* Point object describing the position to test.

**Returns:**

true if position *pt* is within this Rect's area, else false

**6.217.2.4 Rect & CEGUI::Rect::constrainSizeMax (const Size & sz)**

check the size of the Rect object and if it is bigger than *sz*, resize it so it isn't.

**Parameters:**

*sz* Size object that describes the maximum dimensions that this Rect should be limited to.

**Returns:**

'this' Rect object after the constrain operation

**6.217.2.5 Rect & CEGUI::Rect::constrainSizeMin (const Size & sz)**

check the size of the Rect object and if it is smaller than *sz*, resize it so it isn't.

**Parameters:**

*sz* Size object that describes the minimum dimensions that this Rect should be limited to.

**Returns:**

'this' Rect object after the constrain operation

**6.217.2.6 Rect & CEGUI::Rect::constrainSize (const Size & max\_sz, const Size & min\_sz)**

check the size of the Rect object and if it is bigger than *max\_sz* or smaller than *min\_sz*, resize it so it isn't.

**Parameters:**

*max\_sz* Size object that describes the maximum dimensions that this Rect should be limited to.

*min\_sz* Size object that describes the minimum dimensions that this Rect should be limited to.

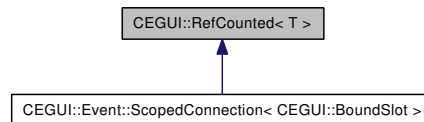
**Returns:**

'this' Rect object after the constrain operation

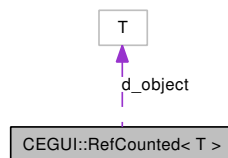
## 6.218 CEGUI::RefCounted< T > Class Template Reference

Simple, generic, reference counted pointer class. This is primarily here for use by the Events system to track when to delete slot bindings.

Inheritance diagram for CEGUI::RefCounted< T >:



Collaboration diagram for CEGUI::RefCounted< T >:



### Public Member Functions

- [RefCounted](#) ()  
*Default constructor.*
- [RefCounted](#) (T \*ob)  
*Construct a [RefCounted](#) object that wraps the pointer ob.*
- [RefCounted](#) (const [RefCounted](#)< T > &other)  
*Copy constructor.*
- [~RefCounted](#) ()  
*Destructor. Only deletes the associated object if no further references exist.*
- [RefCounted](#)< T > & [operator=](#) (const [RefCounted](#)< T > &other)  
*Assignment operator. Previously held object gets its reference count reduced, and is deleted if no further references exist. The newly assigned object, taken from other, gets its count increased.*
- bool [operator==](#) (const [RefCounted](#)< T > &other) const  
*Return whether the two [RefCounted](#) ptrs are equal (point to the same object).*
- bool [operator!=](#) (const [RefCounted](#)< T > &other) const  
*Return whether the two [RefCounted](#) ptrs are not equal (point to different objects).*
- const T & [operator \\*](#) () const  
*Return the object referred to by the wrapped pointer. (beware of null pointers when using this!).*
- T & [operator \\*](#) ()

- `const T * operator → () const`

*Return the wrapped pointer.*

- `T * operator → ()`

- `bool isValid () const`

*Return whether the wrapped pointer is valid. i.e. that it is not null.*

### 6.218.1 Detailed Description

**`template<typename T> class CEGUI::RefCounted< T >`**

Simple, generic, reference counted pointer class. This is primarily here for use by the Events system to track when to delete slot bindings.

## 6.219 CEGUI::RegexValidator Struct Reference

Internal struct to contain compiled regex string.

### Public Member Functions

- void **release** ()

### Public Attributes

- pcre \* **d\_regex**

### 6.219.1 Detailed Description

Internal struct to contain compiled regex string.

## 6.220 CEGUI::RenderCache Class Reference

Class that acts as a cache for images that need to be rendered.

### Public Member Functions

- [RenderCache](#) ()  
*Constructor.*
- [~RenderCache](#) ()  
*Destructor.*
- bool [hasCachedImagery](#) () const  
*Return whether the cache contains anything to draw.*
- void [render](#) (const [Point](#) &basePos, float baseZ, const [Rect](#) &clipper)  
*Send the contents of the cache to the [Renderer](#).*
- void [clearCachedImagery](#) ()  
*Erase any stored image information.*
- void [cacheImage](#) (const [Image](#) &image, const [Rect](#) &destArea, float zOffset, const [ColourRect](#) &cols, const [Rect](#) \*clipper=0, bool clipToDisplay=false)  
*Add an image to the cache.*
- void [cacheText](#) (const [String](#) &text, [Font](#) \*font, [TextFormatting](#) format, const [Rect](#) &destArea, float zOffset, const [ColourRect](#) &cols, const [Rect](#) \*clipper=0, bool clipToDisplay=false)  
*Add a text to the cache.*

### Classes

- struct **ImageInfo**  
*internal struct that holds info about a single image to be drawn.*
- struct **TextInfo**  
*internal struct that holds info about text to be drawn.*

### 6.220.1 Detailed Description

Class that acts as a cache for images that need to be rendered.

This is in many ways an optimisation cache, it allows a full image redraw to occur while limiting the amount of information that needs to be re-calculated.

Basically, unless the actual images (or their size) change, or the colours (or alpha) change then imagery cached in here will suffice for a full redraw. The reasoning behind this is that when some window underneath window 'X' changes, a full image redraw is required by the renderer, however, since window

'X' is unchanged, performing a total recalculation of all imagery is very wasteful, and so we use this cache to limit such waste.

As another example, when a window is simply moved, there is no need to perform a total imagery recalculation; we can still use the imagery cached here since it is position independant.

## 6.220.2 Member Function Documentation

### 6.220.2.1 bool CEGUI::RenderCache::hasCachedImagery () const

Return whether the cache contains anything to draw.

#### Returns:

- true if the cache contains information about images to be drawn.
- false if the cache is empty.

### 6.220.2.2 void CEGUI::RenderCache::render (const Point & *basePos*, float *baseZ*, const Rect & *clipper*)

Send the contents of the cache to the [Renderer](#).

#### Parameters:

*basePos* Point that describes a screen offset that cached imagery will be rendered relative to.

*baseZ* Z value that cached imagery will use as a base figure when calculating final z values.

*clipper* [Rect](#) object describing a rect to which imagery will be clipped.

#### Returns:

Nothing

### 6.220.2.3 void CEGUI::RenderCache::cacheImage (const Image & *image*, const Rect & *destArea*, float *zOffset*, const ColourRect & *cols*, const Rect \* *clipper* = 0, bool *clipToDisplay* = false)

Add an image to the cache.

#### Parameters:

*image* [Image](#) object to be cached.

*destArea* Destination area over which the [Image](#) object will be rendered. This area should be position independant; so position (0,0) will be to top-left corner of whatever it is you're rendering (like a [Window](#) for example).

*zOffset* Zero based z offset for this image. Allows imagery to be layered.

*cols* [ColourRect](#) object describing the colours to be applied when rendering this image.

#### Returns:

Nothing

**6.220.2.4** `void CEGUI::RenderCache::cacheText (const String & text, Font * font, TextFormatting format, const Rect & destArea, float zOffset, const ColourRect & cols, const Rect * clipper = 0, bool clipToDisplay = false)`

Add a text to the cache.

**Parameters:**

*text* [String](#) object to be cached.

*font* [Font](#) to be used when rendering.

*format* TextFormatting value specifying the formatting to use when rendering.

*destArea* Destination area over which the [Image](#) object will be rendered. This area should be position independant; so position (0,0) will be to top-left corner of whatever it is you're rendering (like a [Window](#) for example).

*zOffset* Zero based z offset for this image. Allows imagery to be layered.

*cols* [ColourRect](#) object describing the colours to be applied when rendering this image.

**Returns:**

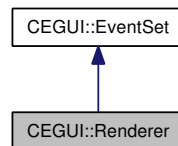
Nothing



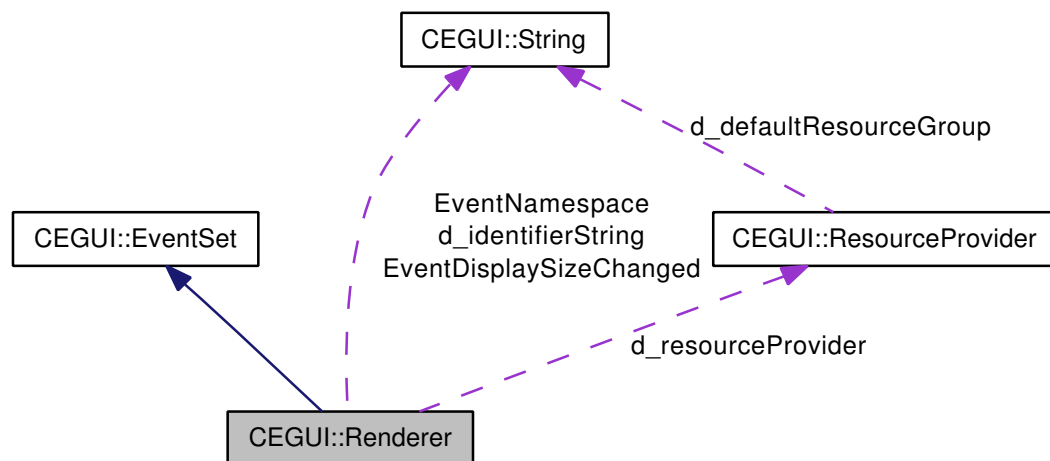
## 6.221 CEGUI::Renderer Class Reference

Abstract class defining the interface for [Renderer](#) objects.

Inheritance diagram for CEGUI::Renderer:



Collaboration diagram for CEGUI::Renderer:



### Public Member Functions

- virtual void [addQuad](#) (const [Rect](#) &dest\_rect, float z, const [Texture](#) \*tex, const [Rect](#) &texture\_rect, const [ColourRect](#) &colours, [QuadSplitMode](#) quad\_split\_mode)=0  
*Add a quad to the rendering queue. All clipping and other adjustments should have been made prior to calling this.*
- virtual void [doRender](#) (void)=0  
*Perform final rendering for all quads that have been queued for rendering.*
- virtual void [clearRenderList](#) (void)=0  
*Clears all queued quads from the render queue.*
- virtual void [setQueueingEnabled](#) (bool setting)=0  
*Enable or disable the queueing of quads from this point on.*
- virtual [Texture](#) \* [createTexture](#) (void)=0  
*Creates a 'null' [Texture](#) object.*
- virtual [Texture](#) \* [createTexture](#) (const [String](#) &filename, const [String](#) &resourceGroup)=0

Create a [Texture](#) object using the given image file.

- virtual [Texture](#) \* [createTexture](#) (float size)=0  
Create a [Texture](#) object with the given pixel dimensions as specified by size. NB: Textures are always square.
- virtual void [destroyTexture](#) ([Texture](#) \*texture)=0  
Destroy the given [Texture](#) object.
- virtual void [destroyAllTextures](#) (void)=0  
Destroy all [Texture](#) objects.
- virtual bool [isQueueingEnabled](#) (void) const =0  
Return whether queueing is enabled.
- virtual float [getWidth](#) (void) const =0  
Return the current width of the display in pixels.
- virtual float [getHeight](#) (void) const =0  
Return the current height of the display in pixels.
- virtual [Size](#) [getSize](#) (void) const =0  
Return the size of the display in pixels.
- virtual [Rect](#) [getRect](#) (void) const =0  
Return a [Rect](#) describing the screen.
- virtual uint [getMaxTextureSize](#) (void) const =0  
Return the maximum texture size available.
- virtual uint [getHorzScreenDPI](#) (void) const =0  
Return the horizontal display resolution dpi.
- virtual uint [getVertScreenDPI](#) (void) const =0  
Return the vertical display resolution dpi.
- void [resetZValue](#) (void)  
Reset the z co-ordinate for rendering.
- void [advanceZValue](#) (void)  
Update the z co-ordinate for the next major UI element (window).
- float [getCurrentZ](#) (void) const  
return the current Z value to use (equates to layer 0 for this UI element).
- float [getZLayer](#) (uint layer) const  
return the z co-ordinate to use for the requested layer on the current GUI element.
- const [String](#) & [getIdentifierString](#) () const  
Return identification string for the renderer module. If the internal id string has not been set by the [Renderer](#) module creator, a generic string of "Unknown renderer" will be returned.
- virtual [ResourceProvider](#) \* [createResourceProvider](#) (void)

## Static Public Attributes

- static const [String EventNamespace](#)  
*Namespace for global events.*
- static const [String EventDisplaySizeChanged](#)

## Protected Attributes

- [ResourceProvider \\* d\\_resourceProvider](#)  
*Holds the pointer to the [ResourceProvider](#) object.*
- [String d\\_identifierString](#)  
*String that holds some id information about the renderer.*

### 6.221.1 Detailed Description

Abstract class defining the interface for [Renderer](#) objects.

Objects derived from [Renderer](#) are the means by which the GUI system interfaces with specific rendering technologies. To use a rendering system or API to draw [CEGUI](#) imagery requires that an appropriate [Renderer](#) object be available.

### 6.221.2 Member Function Documentation

**6.221.2.1** `virtual void CEGUI::Renderer::addQuad (const Rect & dest_rect, float z, const Texture * tex, const Rect & texture_rect, const ColourRect & colours, QuadSplitMode quad_split_mode) [pure virtual]`

Add a quad to the rendering queue. All clipping and other adjustments should have been made prior to calling this.

#### Parameters:

*dest\_rect* [Rect](#) object describing the destination area (values are in pixels)

*z* float value specifying the z co-ordinate / z order of the quad

*tex* pointer to the [Texture](#) object that holds the imagery to be rendered

*texture\_rect* [Rect](#) object holding the area of *tex* that is to be rendered (values are in texture co-ordinates).

*colours* [ColourRect](#) object describing the [colour](#) values that are to be applied when rendering.

*quad\_split\_mode* One of the QuadSplitMode values specifying the way quads are split into triangles

#### Returns:

Nothing

**6.221.2.2 virtual void CEGUI::Renderer::doRender (void) [pure virtual]**

Perform final rendering for all quads that have been queued for rendering.

The contents of the rendering queue is retained and can be rendered again as required. If the contents is not required call [clearRenderList\(\)](#).

**Returns:**

Nothing

**6.221.2.3 virtual void CEGUI::Renderer::clearRenderList (void) [pure virtual]**

Clears all queued quads from the render queue.

**Returns:**

Nothing

**6.221.2.4 virtual void CEGUI::Renderer::setQueueingEnabled (bool *setting*) [pure virtual]**

Enable or disable the queueing of quads from this point on.

This only affects queueing. If queueing is turned off, any calls to `addQuad` will cause the quad to be rendered directly. Note that disabling queueing will not cause currently queued quads to be rendered, nor is the queue cleared - at any time the queue can still be drawn by calling `doRender`, and the list can be cleared by calling `clearRenderList`. Re-enabling the queue causes subsequent quads to be added as if queueing had never been disabled.

**Parameters:**

*setting* true to enable queueing, or false to disable queueing (see notes above).

**Returns:**

Nothing

**6.221.2.5 virtual Texture\* CEGUI::Renderer::createTexture (void) [pure virtual]**

Creates a 'null' [Texture](#) object.

**Returns:**

a newly created [Texture](#) object. The returned [Texture](#) object has no size or imagery associated with it, and is generally of little or no use.

**6.221.2.6 virtual Texture\* CEGUI::Renderer::createTexture (const String & *filename*, const String & *resourceGroup*) [pure virtual]**

Create a [Texture](#) object using the given image file.

**Parameters:**

*filename* [String](#) object that specifies the path and filename of the image file to use when creating the texture.

*resourceGroup* Resource group identifier to be passed to the resource provider when loading the texture file.

**Returns:**

a newly created [Texture](#) object. The initial contents of the texture memory is the requested image file.

**Note:**

Textures are always created with a size that is a power of 2. If the file you specify is of a size that is not a power of two, the final size will be rounded up. Additionally, textures are always square, so the ultimate size is governed by the larger of the width and height of the specified file. You can check the ultimate sizes by querying the texture after creation.

**6.221.2.7 virtual Texture\* CEGUI::Renderer::createTexture (float size) [pure virtual]**

Create a [Texture](#) object with the given pixel dimensions as specified by *size*. NB: Textures are always square.

**Parameters:**

*size* float value that specifies the size to use for the width and height when creating the new texture.

**Returns:**

a newly created [Texture](#) object. The initial contents of the texture memory is undefined / random.

**Note:**

Textures are always created with a size that is a power of 2. If you specify a size that is not a power of two, the final size will be rounded up. So if you specify a size of 1024, the texture will be (1024 x 1024), however, if you specify a size of 1025, the texture will be (2048 x 2048). You can check the ultimate size by querying the texture after creation.

**6.221.2.8 virtual void CEGUI::Renderer::destroyTexture (Texture \* texture) [pure virtual]**

Destroy the given [Texture](#) object.

**Parameters:**

*texture* pointer to the [Texture](#) object to be destroyed

**Returns:**

Nothing

**6.221.2.9 virtual void CEGUI::Renderer::destroyAllTextures (void) [pure virtual]**

Destroy all [Texture](#) objects.

**Returns:**

Nothing

**6.221.2.10 virtual bool CEGUI::Renderer::isQueueingEnabled (void) const** [pure virtual]

Return whether queueing is enabled.

**Returns:**

true if queueing is enabled, false if queueing is disabled.

**6.221.2.11 virtual float CEGUI::Renderer::getWidth (void) const** [pure virtual]

Return the current width of the display in pixels.

**Returns:**

float value equal to the current width of the display in pixels.

**6.221.2.12 virtual float CEGUI::Renderer::getHeight (void) const** [pure virtual]

Return the current height of the display in pixels.

**Returns:**

float value equal to the current height of the display in pixels.

**6.221.2.13 virtual Size CEGUI::Renderer::getSize (void) const** [pure virtual]

Return the size of the display in pixels.

**Returns:**

[Size](#) object describing the dimensions of the current display.

**6.221.2.14 virtual Rect CEGUI::Renderer::getRect (void) const** [pure virtual]

Return a [Rect](#) describing the screen.

**Returns:**

A [Rect](#) object that describes the screen area. Typically, the top-left values are always 0, and the size of the area described is equal to the screen resolution.

**6.221.2.15 virtual uint CEGUI::Renderer::getMaxTextureSize (void) const** [pure virtual]

Return the maximum texture size available.

**Returns:**

[Size](#) of the maximum supported texture in pixels (textures are always assumed to be square)

**6.221.2.16 virtual uint CEGUI::Renderer::getHorzScreenDPI (void) const** [pure virtual]

Return the horizontal display resolution dpi.

**Returns:**

horizontal resolution of the display in dpi.

**6.221.2.17 virtual uint CEGUI::Renderer::getVertScreenDPI (void) const** [pure virtual]

Return the vertical display resolution dpi.

**Returns:**

vertical resolution of the display in dpi.

**6.221.2.18 void CEGUI::Renderer::resetZValue (void)** [inline]

Reset the z co-ordinate for rendering.

**Returns:**

Nothing

**6.221.2.19 void CEGUI::Renderer::advanceZValue (void)** [inline]

Update the z co-ordinate for the next major UI element (window).

**Returns:**

Nothing

**6.221.2.20 float CEGUI::Renderer::getCurrentZ (void) const** [inline]

return the current Z value to use (equates to layer 0 for this UI element).

**Returns:**

float value that specifies the z co-ordinate to be used for layer 0 on the current GUI element.

**6.221.2.21 float CEGUI::Renderer::getZLayer (uint *layer*) const** [inline]

return the z co-ordinate to use for the requested layer on the current GUI element.

**Parameters:**

*layer* Specifies the layer to return the Z co-ordinate for. Each GUI element can use up to 10 layers, so valid inputs are 0 to 9 inclusive. If you specify an invalid value for *layer*, results are undefined.

**Returns:**

float value that specifies the Z co-ordinate for layer *layer* on the current GUI element.

**6.221.2.22   const String & CEGUI::Renderer::getIdentifierString () const**

Return identification string for the renderer module. If the internal id string has not been set by the [Renderer](#) module creator, a generic string of "Unknown renderer" will be returned.

**Returns:**

[String](#) object holding a string that identifies the [Renderer](#) in use.

**6.221.3   Member Data Documentation****6.221.3.1   const String CEGUI::Renderer::EventDisplaySizeChanged   [static]**

event that fires when the underlying display size had changed.

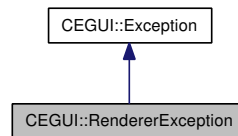
It is important that all [Renderer](#) implementers fire this properly as the system itself subscribes to this event.



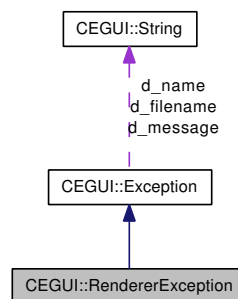
## 6.222 CEGUI::RendererException Class Reference

**Exception** class used when an problem is detected within the **Renderer** or related objects.

Inheritance diagram for CEGUI::RendererException:



Collaboration diagram for CEGUI::RendererException:



### Public Member Functions

- **RendererException** (const **String** &message, const **String** &file="unknown", int line=0)  
*Constructor that is responsible for logging the renderer exception by calling the base class.*

#### 6.222.1 Detailed Description

**Exception** class used when an problem is detected within the **Renderer** or related objects.

#### 6.222.2 Constructor & Destructor Documentation

**6.222.2.1 CEGUI::RendererException::RendererException (const **String** & message, const **String** & file = "unknown", int line = 0) [inline]**

Constructor that is responsible for logging the renderer exception by calling the base class.

#### Parameters:

- message** **String** object describing the reason for the renderer exception being thrown.
- filename** **String** object containing the name of the file where the renderer exception occurred.
- line** Integer representing the line number where the renderer exception occurred.

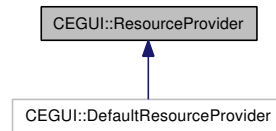
**Remarks:**

The renderer exception name is automatically passed to the base class as "CEGUI::RendererException".

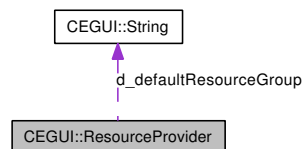
## 6.223 CEGUI::ResourceProvider Class Reference

Abstract class that defines the required interface for all resource provider sub-classes.

Inheritance diagram for CEGUI::ResourceProvider:



Collaboration diagram for CEGUI::ResourceProvider:



### Public Member Functions

- [ResourceProvider](#) ()  
*Constructor for the [ResourceProvider](#) class.*
- virtual [~ResourceProvider](#) (void)  
*Destructor for the [ResourceProvider](#) class.*
- virtual void [loadRawDataContainer](#) (const [String](#) &filename, [RawDataContainer](#) &output, const [String](#) &resourceGroup)=0  
*Load XML data using [InputSource](#) objects.*
- virtual void [unloadRawDataContainer](#) ([RawDataContainer](#) &data)  
*Unload raw binary data. This gives the resource provider a change to unload the data in its own way before the data container object is destroyed. If it does nothing, then the object will release its memory.*
- const [String](#) & [getDefaultResourceGroup](#) (void) const  
*Return the current default resource group identifier.*
- void [setDefaultResourceGroup](#) (const [String](#) &resourceGroup)  
*Set the default resource group identifier.*

### Protected Attributes

- [String](#) d\_defaultResourceGroup  
*Default resource group identifier.*

### 6.223.1 Detailed Description

Abstract class that defines the required interface for all resource provider sub-classes.

A [ResourceProvider](#) is used to load both XML and binary data from an external source. This could be from a filesystem or the resource manager of a specific renderer.

### 6.223.2 Member Function Documentation

**6.223.2.1** `virtual void CEGUI::ResourceProvider::loadRawDataContainer (const String & filename, RawDataContainer & output, const String & resourceGroup) [pure virtual]`

Load XML data using InputSource objects.

**Parameters:**

*filename* [String](#) containing a filename of the resource to be loaded.

*output* Reference to a InputSourceContainer object to load the data into.

Load raw binary data.

**Parameters:**

*filename* [String](#) containing a filename of the resource to be loaded.

*output* Reference to a [RawDataContainer](#) object to load the data into.

*resourceGroup* Optional [String](#) that may be used by implementations to identify the group from which the resource should be loaded.

**6.223.2.2** `virtual void CEGUI::ResourceProvider::unloadRawDataContainer (RawDataContainer & data) [inline, virtual]`

Unload raw binary data. This gives the resource provider a change to unload the data in its own way before the data container object is destroyed. If it does nothing, then the object will release its memory.

**Parameters:**

*data* Reference to a [RawDataContainer](#) object that is about to be destroyed.

**6.223.2.3** `const String& CEGUI::ResourceProvider::getDefaultResourceGroup (void) const [inline]`

Return the current default resource group identifier.

**Returns:**

[String](#) object containing the currently set default resource group identifier.

#### 6.223.2.4 void CEGUI::ResourceProvider::setDefaultResourceGroup (const String & *resourceGroup*) [inline]

Set the default resource group identifier.

##### Parameters:

*resourceGroup* [String](#) object containing the default resource group identifier to be used.

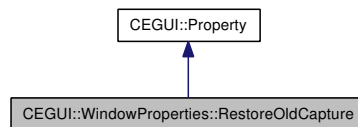
##### Returns:

Nothing.

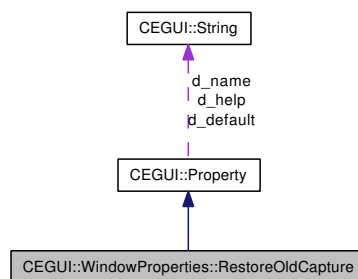
## 6.224 CEGUI::WindowProperties::RestoreOldCapture Class Reference

[Property](#) to access window Restore Old Capture setting.

Inheritance diagram for CEGUI::WindowProperties::RestoreOldCapture:



Collaboration diagram for CEGUI::WindowProperties::RestoreOldCapture:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.224.1 Detailed Description

[Property](#) to access window Restore Old Capture setting.

This property offers access to the restore old capture setting for the window. This setting is of generally limited use, its primary purpose is for certain operations required for compound widgets.

#### Usage:

- Name: [RestoreOldCapture](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the [Window](#) should restore any previous capture [Window](#) when it loses input capture.
- "False" to indicate the [Window](#) should not restore the old capture [Window](#). This is the default behaviour.

## 6.224.2 Member Function Documentation

### 6.224.2.1 String CEGUI::WindowProperties::RestoreOldCapture::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.224.2.2 void CEGUI::WindowProperties::RestoreOldCapture::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

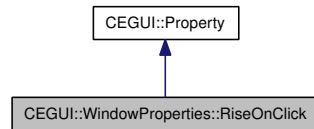
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

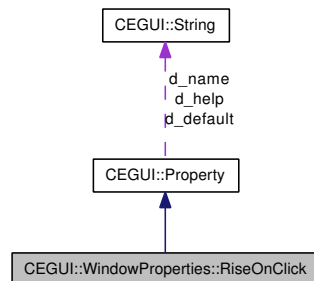
## 6.225 CEGUI::WindowProperties::RiseOnClick Class Reference

[Property](#) to access whether the window rises to the top of the z order when clicked.

Inheritance diagram for CEGUI::WindowProperties::RiseOnClick:



Collaboration diagram for CEGUI::WindowProperties::RiseOnClick:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.225.1 Detailed Description

[Property](#) to access whether the window rises to the top of the z order when clicked.

Usage:

- Name: [RiseOnClick](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the [Window](#) will rise to the surface when clicked.
- "False" to indicate the [Window](#) will not change z position when clicked.



## 6.225.2 Member Function Documentation

### 6.225.2.1 String CEGUI::WindowProperties::RiseOnClick::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.225.2.2 void CEGUI::WindowProperties::RiseOnClick::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

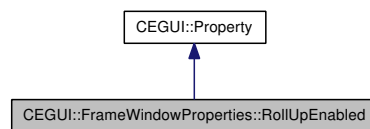
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

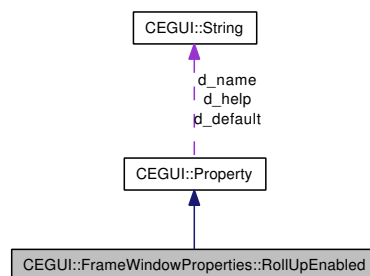
## 6.226 CEGUI::FrameWindowProperties::RollUpEnabled Class Reference

[Property](#) to access the setting for whether the user is able to roll-up / shade the window.

Inheritance diagram for CEGUI::FrameWindowProperties::RollUpEnabled:



Collaboration diagram for CEGUI::FrameWindowProperties::RollUpEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.226.1 Detailed Description

[Property](#) to access the setting for whether the user is able to roll-up / shade the window.

#### Usage:

- Name: [RollUpEnabled](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the user can roll-up / shade the window.
- "False" to indicate the user can not roll-up / shade the window.

## 6.226.2 Member Function Documentation

### 6.226.2.1 String CEGUI::FrameWindowProperties::RollUpEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.226.2.2 void CEGUI::FrameWindowProperties::RollUpEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

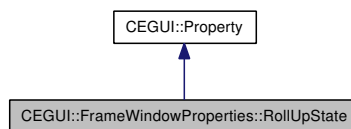
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

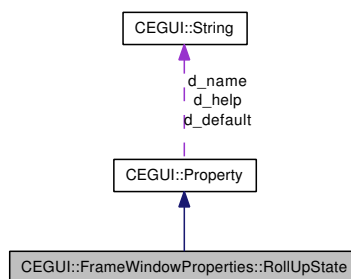
## 6.227 CEGUI::FrameWindowProperties::RollUpState Class Reference

[Property](#) to access the roll-up / shade state of the window.

Inheritance diagram for CEGUI::FrameWindowProperties::RollUpState:



Collaboration diagram for CEGUI::FrameWindowProperties::RollUpState:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.227.1 Detailed Description

[Property](#) to access the roll-up / shade state of the window.

**Usage:**

- Name: [RollUpState](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the window is / should be rolled-up.
- "False" to indicate the window is not / should not be rolled up

## 6.227.2 Member Function Documentation

### 6.227.2.1 String CEGUI::FrameWindowProperties::RollUpState::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.227.2.2 void CEGUI::FrameWindowProperties::RollUpState::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

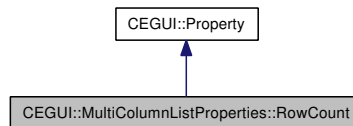
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

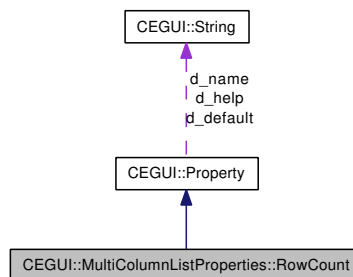
## 6.228 CEGUI::MultiColumnListProperties::RowCount Class Reference

[Property](#) to access the number of rows in the list (read-only).

Inheritance diagram for CEGUI::MultiColumnListProperties::RowCount:



Collaboration diagram for CEGUI::MultiColumnListProperties::RowCount:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.228.1 Detailed Description

[Property](#) to access the number of rows in the list (read-only).

Usage:

- Name: [RowCount](#)
- Format: "" (property is read-only).

### 6.228.2 Member Function Documentation

#### 6.228.2.1 String CEGUI::MultiColumnListProperties::RowCount::get (const [PropertyReceiver](#) \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.228.2.2** `void CEGUI::MultiColumnListProperties::RowCount::set (PropertyReceiver * receiver,  
const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

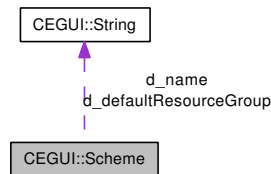
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.229 CEGUI::Scheme Class Reference

A class that groups a set of GUI elements and initialises the system to access those elements.

Collaboration diagram for CEGUI::Scheme:



### Public Member Functions

- void [loadResources](#) (void)  
*Loads all resources for this scheme.*
- void [unloadResources](#) (void)  
*Unloads all resources for this scheme. This should be used very carefully.*
- bool [resourcesLoaded](#) (void) const  
*Return whether the resources for this *Scheme* are all loaded.*
- const [String](#) & [getName](#) (void) const  
*Return the name of this *Scheme*.*
- [~Scheme](#) (void)  
*Destroys a *Scheme* object.*

### Static Public Member Functions

- static const [String](#) & [getDefaultResourceGroup](#) ()  
*Returns the default resource group currently set for Schemes.*
- static void [setDefaultResourceGroup](#) (const [String](#) &resourceGroup)  
*Sets the default resource group to be used when loading scheme xml data.*

### Friends

- class [Scheme\\_xmlHandler](#)
- [Scheme](#) \* [SchemeManager::loadScheme](#) (const [String](#) &scheme\_filename, const [String](#) &resourceGroup)
- void [SchemeManager::unloadScheme](#) (const [String](#) &scheme\_name)



## Classes

- struct **AliasMapping**
- struct **FalagardMapping**
- struct **LoadableUIElement**
- struct **UIElementFactory**
- struct **UIModule**

### 6.229.1 Detailed Description

A class that groups a set of GUI elements and initialises the system to access those elements.

A GUI [Scheme](#) is a high-level construct that loads and initialises various lower-level objects and registers them within the system for usage. So, for example, a [Scheme](#) might create some [Imageset](#) objects, some [Font](#) objects, and register a collection of [WindowFactory](#) objects within the system which would then be in a state to serve those elements to client code.

### 6.229.2 Constructor & Destructor Documentation

#### 6.229.2.1 CEGUI::Scheme::~~Scheme (void)

Destroys a [Scheme](#) object.

##### Returns:

Nothing

### 6.229.3 Member Function Documentation

#### 6.229.3.1 void CEGUI::Scheme::loadResources (void)

Loads all resources for this scheme.

##### Returns:

Nothing.

#### 6.229.3.2 void CEGUI::Scheme::unloadResources (void)

Unloads all resources for this scheme. This should be used very carefully.

##### Returns:

Nothing.

#### 6.229.3.3 bool CEGUI::Scheme::resourcesLoaded (void) const

Return whether the resources for this [Scheme](#) are all loaded.

**Returns:**

true if all resources for the [Scheme](#) are loaded and available, or false if one or more resource is not currently loaded.

**6.229.3.4** `const String& CEGUI::Scheme::getName (void) const` [inline]

Return the name of this [Scheme](#).

**Returns:**

[String](#) object containing the name of this [Scheme](#).

**6.229.3.5** `static const String& CEGUI::Scheme::getDefaultResourceGroup (void)` [inline, static]

Returns the default resource group currently set for Schemes.

**Returns:**

[String](#) describing the default resource group identifier that will be used when loading [Scheme](#) xml file data.

**6.229.3.6** `static void CEGUI::Scheme::setDefaultResourceGroup (const String & resourceGroup)` [inline, static]

Sets the default resource group to be used when loading scheme xml data.

**Parameters:**

*resourceGroup* [String](#) describing the default resource group identifier to be used.

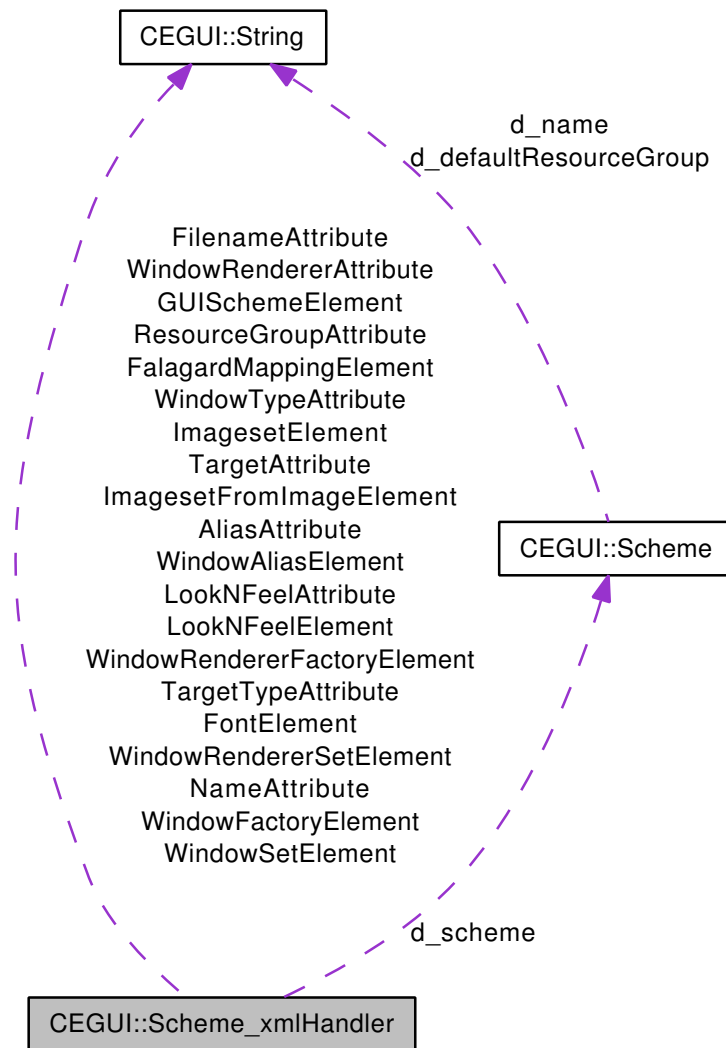
**Returns:**

Nothing.

## 6.230 CEGUI::Scheme\_xmlHandler Class Reference

Handler class used to parse the [Scheme](#) XML files using SAX2.

Collaboration diagram for CEGUI::Scheme\_xmlHandler:



### Public Member Functions

- [Scheme\\_xmlHandler](#) ([Scheme](#) \*scheme)  
*Constructor for Scheme::xmlHandler objects.*
- virtual [~Scheme\\_xmlHandler](#) (void)  
*Destructor for Scheme::xmlHandler objects.*
- virtual void [elementStart](#) (const [String](#) &element, const [XMLAttributes](#) &attributes)  
*document processing (only care about elements, schema validates format)*

- virtual void **elementEnd** (const [String](#) &element)

### 6.230.1 Detailed Description

Handler class used to parse the [Scheme](#) XML files using SAX2.

### 6.230.2 Constructor & Destructor Documentation

#### 6.230.2.1 CEGUI::Scheme\_xmlHandler::Scheme\_xmlHandler (Scheme \* *scheme*) [inline]

Constructor for Scheme::xmlHandler objects.

#### Parameters:

*scheme* Pointer to the [Scheme](#) object creating this xmlHandler object

## 6.231 CEGUI::SchemeManager Class Reference

A class that manages the creation of, access to, and destruction of GUI [Scheme](#) objects.

### Public Types

- typedef [ConstBaseIterator](#)< SchemeRegistry > **SchemeIterator**

### Public Member Functions

- [SchemeManager](#) (void)  
*Constructor for [SchemeManager](#) objects.*
- [~SchemeManager](#) (void)  
*Destructor for [SchemeManager](#) objects.*
- [Scheme](#) \* [loadScheme](#) (const [String](#) &scheme\_filename, const [String](#) &resourceGroup="")  
*Loads a scheme.*
- void [unloadScheme](#) (const [String](#) &scheme\_name)  
*Unloads all data referenced in a scheme. If any object is using some resource which is listed in the scheme, this function will effectively pull the rug out from under those objects. This should be used with extreme caution, or not at all.*
- bool [isSchemePresent](#) (const [String](#) &scheme\_name) const  
*Returns true if the named [Scheme](#) is present in the system (though the resources for the scheme may or may not be loaded).*
- [Scheme](#) \* [getScheme](#) (const [String](#) &name) const  
*Returns a pointer to the [Scheme](#) object with the specified name.*
- void [unloadAllSchemes](#) (void)  
*Unload all schemes currently defined within the system.*
- [SchemeIterator](#) [getIterator](#) (void) const  
*Return a [SchemeManager::SchemeIterator](#) object to iterate over the available schemes.*

### 6.231.1 Detailed Description

A class that manages the creation of, access to, and destruction of GUI [Scheme](#) objects.

### 6.231.2 Member Function Documentation

#### 6.231.2.1 [Scheme](#) \* CEGUI::SchemeManager::loadScheme (const [String](#) & scheme\_filename, const [String](#) & resourceGroup = "")

Loads a scheme.

**Parameters:**

*scheme\_filename* [String](#) object that holds the filename of the scheme to be loaded

*resourceGroup* Resource group identifier to be passed to the resource manager. NB: This affects loading of the scheme xml file only, scheme resources may specify their own groups.

**Returns:**

Pointer to an object representing the loaded [Scheme](#).

**6.231.2.2 void CEGUI::SchemeManager::unloadScheme (const String & scheme\_name)**

Unloads all data referenced in a scheme. If any object is using some resource which is listed in the scheme, this function will effectively pull the rug out from under those objects. This should be used with extreme caution, or not at all.

**Parameters:**

*scheme\_name* [String](#) object specifying the name of the [Scheme](#) to be unloaded.

**6.231.2.3 bool CEGUI::SchemeManager::isSchemePresent (const String & scheme\_name) const**  
[inline]

Returns true if the named [Scheme](#) is present in the system (though the resources for the scheme may or may not be loaded).

**Parameters:**

*scheme\_name* [String](#) object specifying the name of the [Scheme](#) to check for.

**Returns:**

true if the scheme is loaded, false if it is not.

**6.231.2.4 Scheme \* CEGUI::SchemeManager::getScheme (const String & name) const**

Returns a pointer to the [Scheme](#) object with the specified name.

**Parameters:**

*name* [String](#) object holding the name of the [Scheme](#) to be returned.

**Returns:**

Pointer to the [Scheme](#) named *name*.

**Exceptions:**

[UnknownObjectException](#) thrown if no [Scheme](#) named *name* is present in the system

**6.231.2.5 void CEGUI::SchemeManager::unloadAllSchemes (void)**

Unload all schemes currently defined within the system.

**Note:**

Calling this method has the potential to be very dangerous; if any of the data that forms part of the scheme is still in use, you can expect fireworks shortly after!

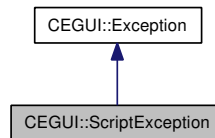
**Returns:**

Nothing.

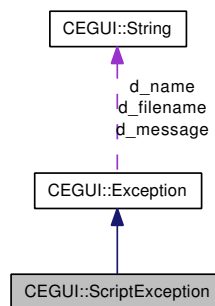
## 6.232 CEGUI::ScriptException Class Reference

[Exception](#) class used when a scripting error occurs.

Inheritance diagram for CEGUI::ScriptException:



Collaboration diagram for CEGUI::ScriptException:



### Public Member Functions

- [ScriptException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the script exception by calling the base class.*

### 6.232.1 Detailed Description

[Exception](#) class used when a scripting error occurs.

### 6.232.2 Constructor & Destructor Documentation

#### 6.232.2.1 CEGUI::ScriptException::ScriptException (const [String](#) & message, const [String](#) & file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the script exception by calling the base class.

#### Parameters:

- message* [String](#) object describing the reason for the script exception being thrown.
- filename* [String](#) object containing the name of the file where the script exception occurred.
- line* Integer representing the line number where the script exception occurred.



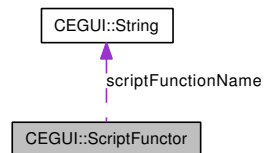
**Remarks:**

The script exception name is automatically passed to the base class as "CEGUI::ScriptException".

## 6.233 CEGUI::ScriptFunctor Class Reference

Functor class used for binding named script functions to events.

Collaboration diagram for CEGUI::ScriptFunctor:



### Public Member Functions

- **ScriptFunctor** (const [String](#) &functionName)
- **ScriptFunctor** (const [ScriptFunctor](#) &obj)
- **bool operator()** (const [EventArgs](#) &e) const

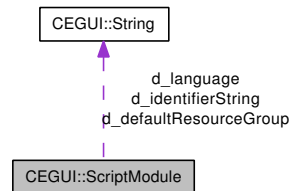
### 6.233.1 Detailed Description

Functor class used for binding named script functions to events.

## 6.234 CEGUI::ScriptModule Class Reference

Abstract interface required for all scripting support modules to be used with the [CEGUI](#) system.

Collaboration diagram for CEGUI::ScriptModule:



### Public Member Functions

- [ScriptModule](#) (void)  
*Constructor for [ScriptModule](#) base class.*
- virtual [~ScriptModule](#) (void)  
*Destructor for [ScriptModule](#) base class.*
- virtual void [executeScriptFile](#) (const [String](#) &filename, const [String](#) &resourceGroup="")=0  
*Execute a script file.*
- virtual int [executeScriptGlobal](#) (const [String](#) &function\_name)=0  
*Execute a scripted global function. The function should not take any parameters and should return an integer.*
- virtual bool [executeScriptedEventHandler](#) (const [String](#) &handler\_name, const [EventArgs](#) &e)=0  
*Execute a scripted global 'event handler' function. The function should take some kind of [EventArgs](#) like parameter that the concrete implementation of this function can create from the passed [EventArgs](#) based object. The function should not return anything.*
- virtual void [executeString](#) (const [String](#) &str)=0  
*Execute script code contained in the given [CEGUI::String](#) object.*
- virtual void [createBindings](#) (void)  
*Method called during system initialisation, prior to running any scripts via the [ScriptModule](#), to enable the [ScriptModule](#) to perform any operations required to complete initialisation or binding of the script language to the gui system objects.*
- virtual void [destroyBindings](#) (void)  
*Method called during system destruction, after all scripts have been run via the [ScriptModule](#), to enable the [ScriptModule](#) to perform any operations required to cleanup bindings of the script language to the gui system objects, as set-up in the earlier [createBindings](#) call.*
- const [String](#) & [getIdentifierString](#) () const  
*Return identification string for the [ScriptModule](#). If the internal id string has not been set by the [ScriptModule](#) creator, a generic string of "Unknown scripting module" will be returned.*

- const [String](#) & [getLanguage](#) () const  
*Return an string which identifies the language of this module. /return [String](#) object holding a string that identifies the language of the module.*
- virtual [Event::Connection](#) [subscribeEvent](#) ([EventSet](#) \*target, const [String](#) &name, const [String](#) &subscriber\_name)=0  
*Subscribes the named [Event](#) to a scripted funtion.*
- virtual [Event::Connection](#) [subscribeEvent](#) ([EventSet](#) \*target, const [String](#) &name, [Event::Group](#) group, const [String](#) &subscriber\_name)=0  
*Subscribes the specified group of the named [Event](#) to a scripted funtion.*

## Static Public Member Functions

- static void [setDefaultResourceGroup](#) (const [String](#) &resourceGroup)  
*Sets the default resource group to be used when loading script files.*
- static const [String](#) & [getDefaultResourceGroup](#) ()  
*Returns the default resource group used when loading script files.*

## Protected Attributes

- [String](#) d\_identifierString  
*[String](#) that holds some id information about the module.*
- [String](#) d\_language  
*[String](#) that holds a string containing the language of the scripting modue.*

## Static Protected Attributes

- static [String](#) d\_defaultResourceGroup  
*holds the default resource group ID for loading script files.*

### 6.234.1 Detailed Description

Abstract interface required for all scripting support modules to be used with the [CEGUI](#) system.

### 6.234.2 Member Function Documentation

- 6.234.2.1** virtual void [CEGUI::ScriptModule::executeScriptFile](#) (const [String](#) &filename, const [String](#) &resourceGroup = "") [pure virtual]

Execute a script file.

**Parameters:**

- filename* [String](#) object holding the filename of the script file that is to be executed
- resourceGroup* Resource group identifier to be passed to the [ResourceProvider](#) when loading the script file.

**6.234.2.2 virtual int CEGUI::ScriptModule::executeScriptGlobal (const String &function\_name)**  
[pure virtual]

Execute a scripted global function. The function should not take any parameters and should return an integer.

**Parameters:**

- function\_name* [String](#) object holding the name of the function, in the global script environment, that is to be executed.

**Returns:**

The integer value returned from the script function.

**6.234.2.3 virtual bool CEGUI::ScriptModule::executeScriptedEventHandler (const String &handler\_name, const EventArgs &e)** [pure virtual]

Execute a scripted global 'event handler' function. The function should take some kind of [EventArgs](#) like parameter that the concrete implementation of this function can create from the passed [EventArgs](#) based object. The function should not return anything.

**Parameters:**

- handler\_name* [String](#) object holding the name of the scripted handler function.
- e* [EventArgs](#) based object that should be passed, by any appropriate means, to the scripted function.

**Returns:**

- true if the event was handled.
- false if the event was not handled.

**6.234.2.4 virtual void CEGUI::ScriptModule::executeString (const String &str)** [pure virtual]

Execute script code contained in the given [CEGUI::String](#) object.

**Parameters:**

- str* [String](#) object holding the valid script code that should be executed.

**Returns:**

Nothing.

**6.234.2.5 virtual void CEGUI::ScriptModule::createBindings (void)** [inline, virtual]

Method called during system initialisation, prior to running any scripts via the [ScriptModule](#), to enable the [ScriptModule](#) to perform any operations required to complete initialisation or binding of the script language to the gui system objects.

**Returns:**

Nothing.

**6.234.2.6 virtual void CEGUI::ScriptModule::destroyBindings (void)** [inline, virtual]

Method called during system destruction, after all scripts have been run via the [ScriptModule](#), to enable the [ScriptModule](#) to perform any operations required to cleanup bindings of the script language to the gui system objects, as set-up in the earlier createBindings call.

**Returns:**

Nothing.

**6.234.2.7 const String & CEGUI::ScriptModule::getIdentifierString () const**

Return identification string for the [ScriptModule](#). If the internal id string has not been set by the [ScriptModule](#) creator, a generic string of "Unknown scripting module" will be returned.

**Returns:**

[String](#) object holding a string that identifies the [ScriptModule](#) in use.

**6.234.2.8 virtual Event::Connection CEGUI::ScriptModule::subscribeEvent (EventSet \* target, const String & name, const String & subscriber\_name)** [pure virtual]

Subscribes the named [Event](#) to a scripted funtion.

**Parameters:**

*target* The target [EventSet](#) for the subscription.

*name* [String](#) object containing the name of the [Event](#) to subscribe to.

*subscriber\_name* [String](#) object containing the name of the script funtion that is to be subscribed to the [Event](#).

**Returns:**

Connection object that can be used to check the status of the [Event](#) connection and to disconnect (unsubscribe) from the [Event](#).

**Exceptions:**

[UnknownObjectException](#) Thrown if an [Event](#) named *name* is not in the [EventSet](#)

**6.234.2.9** `virtual Event::Connection CEGUI::ScriptModule::subscribeEvent (EventSet * target, const String & name, Event::Group group, const String & subscriber_name)` [pure virtual]

Subscribes the specified group of the named [Event](#) to a scripted funtion.

**Parameters:**

*target* The target [EventSet](#) for the subscription.

*name* [String](#) object containing the name of the [Event](#) to subscribe to.

*group* Group which is to be subscribed to. Subscription groups are called in ascending order.

*subscriber\_name* [String](#) object containing the name of the script funtion that is to be subscribed to the [Event](#).

**Returns:**

Connection object that can be used to check the status of the [Event](#) connection and to disconnect (unsubscribe) from the [Event](#).

**Exceptions:**

[UnknownObjectException](#) Thrown if an [Event](#) named *name* is not in the [EventSet](#)

**6.234.2.10** `static void CEGUI::ScriptModule::setDefaultResourceGroup (const String & resourceGroup)` [inline, static]

Sets the default resource group to be used when loading script files.

**Parameters:**

*resourceGroup* [String](#) describing the default resource group identifier to be used.

**Returns:**

Nothing.

**6.234.2.11** `static const String& CEGUI::ScriptModule::getDefaultResourceGroup (void)` [inline, static]

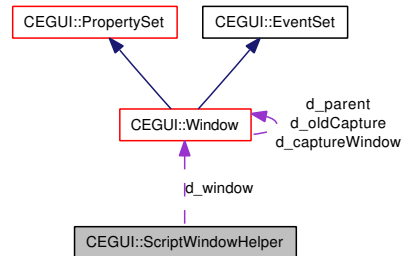
Returns the default resource group used when loading script files.

**Returns:**

[String](#) describing the default resource group identifier..

## 6.235 CEGUI::ScriptWindowHelper Class Reference

Collaboration diagram for CEGUI::ScriptWindowHelper:



### Public Member Functions

- **ScriptWindowHelper** ([Window](#) \*wnd)
- [Window](#) \* [getWindow](#) ()
- [Window](#) \* [getWindow](#) ([String](#) &name)

### Protected Attributes

- [Window](#) \* [d\\_window](#)  
*[Window](#) that is being aided by this helper.*

### 6.235.1 Detailed Description

#### requirements

winXP or later

#### Remarks:

Used by Scriptmodules to help access certain functions or special objects

#### Version:

1.0 first version

#### Date:

05-13-2007

#### Author:

jwelch

#### license

Copyright 2007 Jon Welch & [CEGUI](#) Team

#### [Todo](#)



[Bug](#)

## 6.235.2 Member Function Documentation

### 6.235.2.1 Window \* CEGUI::ScriptWindowHelper::getWindow ()

Returns a pointer to the window that this helper was created for.

**Returns:**

[Window](#) object to helped window

### 6.235.2.2 Window \* CEGUI::ScriptWindowHelper::getWindow (String & *name*)

Returns a pointer to the window object specified using the prefix of the helped window.

**Parameters:**

*name* name of the window you wish to retrieve

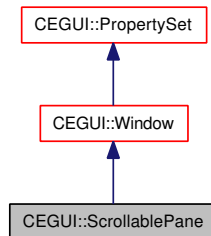
**Returns:**

[Window](#) object to found window, or NULL

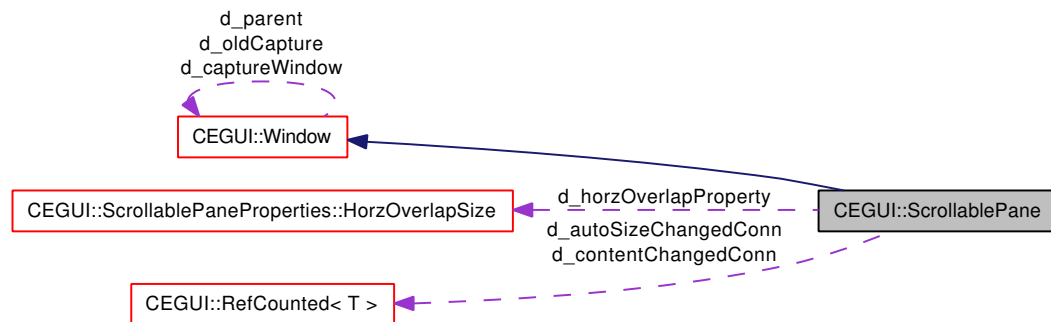
## 6.236 CEGUI::ScrollablePane Class Reference

Base class for the [ScrollablePane](#) widget.

Inheritance diagram for CEGUI::ScrollablePane:



Collaboration diagram for CEGUI::ScrollablePane:



### Public Member Functions

- [ScrollablePane](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for the [ScrollablePane](#) base class.*
- [~ScrollablePane](#) (void)  
*Destructor for the [ScrollablePane](#) base class.*
- const [ScrolledContainer](#) \* [getContentPane](#) (void) const  
*Returns a pointer to the window holding the pane contents.*
- bool [isVertScrollbarAlwaysShown](#) (void) const  
*Return whether the vertical scroll bar is always shown.*
- void [setShowVertScrollbar](#) (bool setting)  
*Set whether the vertical scroll bar should always be shown.*
- bool [isHorzScrollbarAlwaysShown](#) (void) const  
*Return whether the horizontal scroll bar is always shown.*
- void [setShowHorzScrollbar](#) (bool setting)

*Set whether the horizontal scroll bar should always be shown.*

- bool [isContentPaneAutoSized](#) (void) const  
*Return whether the content pane is auto sized.*
- void [setContentPaneAutoSized](#) (bool setting)  
*Set whether the content pane should be auto-sized.*
- const [Rect](#) & [getContentPaneArea](#) (void) const  
*Return the current content pane area for the [ScrollablePane](#).*
- void [setContentPaneArea](#) (const [Rect](#) &area)  
*Set the current content pane area for the [ScrollablePane](#).*
- float [getHorizontalStepSize](#) (void) const  
*Returns the horizontal scrollbar step size as a fraction of one complete view page.*
- void [setHorizontalStepSize](#) (float step)  
*Sets the horizontal scrollbar step size as a fraction of one complete view page.*
- float [getHorizontalOverlapSize](#) (void) const  
*Returns the horizontal scrollbar overlap size as a fraction of one complete view page.*
- void [setHorizontalOverlapSize](#) (float overlap)  
*Sets the horizontal scrollbar overlap size as a fraction of one complete view page.*
- float [getHorizontalScrollPosition](#) (void) const  
*Returns the horizontal scroll position as a fraction of the complete scrollable width.*
- void [setHorizontalScrollPosition](#) (float position)  
*Sets the horizontal scroll position as a fraction of the complete scrollable width.*
- float [getVerticalStepSize](#) (void) const  
*Returns the vertical scrollbar step size as a fraction of one complete view page.*
- void [setVerticalStepSize](#) (float step)  
*Sets the vertical scrollbar step size as a fraction of one complete view page.*
- float [getVerticalOverlapSize](#) (void) const  
*Returns the vertical scrollbar overlap size as a fraction of one complete view page.*
- void [setVerticalOverlapSize](#) (float overlap)  
*Sets the vertical scrollbar overlap size as a fraction of one complete view page.*
- float [getVerticalScrollPosition](#) (void) const  
*Returns the vertical scroll position as a fraction of the complete scrollable height.*
- void [setVerticalScrollPosition](#) (float position)  
*Sets the vertical scroll position as a fraction of the complete scrollable height.*

- [Rect getViewableArea](#) (void) const  
*Return a [Rect](#) that described the pane's viewable area, relative to this [Window](#), in pixels.*
- [Scrollbar \\* getVertScrollbar](#) () const  
*Return a pointer to the vertical scrollbar component widget for this [ScrollablePane](#).*
- [Scrollbar \\* getHorzScrollbar](#) () const  
*Return a pointer to the horizontal scrollbar component widget for this [ScrollablePane](#).*
- void [initialiseComponents](#) (void)  
*Initialises the [Window](#) based object ready for use.*
- void [destroy](#) (void)  
*Internal destroy method which actually just adds the window and any parent destructed child windows to the dead pool.*

## Static Public Attributes

- static const [String WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String EventNamespace](#)  
*Namespace for global events.*
- static const [String EventContentPaneChanged](#)  
*[Event](#) fired when an area on the content pane has been updated.*
- static const [String EventVertScrollbarModeChanged](#)  
*[Event](#) triggered when the vertical scroll bar 'force' setting changes.*
- static const [String EventHorzScrollbarModeChanged](#)  
*[Event](#) triggered when the horizontal scroll bar 'force' setting changes.*
- static const [String EventAutoSizeSettingChanged](#)  
*[Event](#) fired when the auto size setting changes.*
- static const [String EventContentPaneScrolled](#)  
*[Event](#) fired when the pane gets scrolled.*
- static const [String VertScrollbarNameSuffix](#)  
*Widget name suffix for the vertical scrollbar component.*
- static const [String HorzScrollbarNameSuffix](#)  
*Widget name suffix for the horizontal scrollbar component.*
- static const [String ScrolledContainerNameSuffix](#)  
*Widget name suffix for the scrolled container component.*

## Protected Member Functions

- void [configureScrollbars](#) (void)  
*Return a [Rect](#) that described the pane's viewable area, relative to this [Window](#), in pixels.*
- bool [isVertScrollbarNeeded](#) (void) const  
*Return whether the vertical scrollbar is needed.*
- bool [isHorzScrollbarNeeded](#) (void) const  
*Return whether the horizontal scrollbar is needed.*
- void [updateContainerPosition](#) (void)  
*Update the content container position according to the current state of the widget (like scrollbar positions, etc).*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- [ScrolledContainer](#) \* [getScrolledContainer](#) () const  
*Return a pointer to the [ScrolledContainer](#) component widget for this [ScrollablePane](#).*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- virtual void [onContentPaneChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) trigger method called when some pane content has changed size or location.*
- virtual void [onVertScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) trigger method called when the setting that controls whether the vertical scrollbar is always shown or not, is changed.*
- virtual void [onHorzScrollbarModeChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) trigger method called when the setting that controls whether the horizontal scrollbar is always shown or not, is changed.*
- virtual void [onAutoSizeSettingChanged](#) ([WindowEventArgs](#) &e)  
*Notification method called whenever the setting that controls whether the content pane is automatically sized is changed.*
- virtual void [onContentPaneScrolled](#) ([WindowEventArgs](#) &e)  
*Notification method called whenever the content pane is scrolled via changes in the scrollbar positions.*
- bool [handleScrollChange](#) (const [EventArgs](#) &e)  
*Handler method which gets subscribed to the scrollbar position change events.*
- bool [handleContentAreaChange](#) (const [EventArgs](#) &e)  
*Handler method which gets subscribed to the [ScrolledContainer](#) content change events.*
- bool [handleAutoSizePaneChanged](#) (const [EventArgs](#) &e)  
*Handler method which gets subscribed to the [ScrolledContainer](#) auto-size setting changes.*

- void [addChild\\_impl](#) ([Window](#) \*wnd)  
*Add given window to child list at an appropriate position.*
- void [removeChild\\_impl](#) ([Window](#) \*wnd)  
*Remove given window from child list.*
- void [onSized](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's size changes.*
- void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*

## Protected Attributes

- bool [d\\_forceVertScroll](#)  
*true if vertical scrollbar should always be displayed*
- bool [d\\_forceHorzScroll](#)  
*true if horizontal scrollbar should always be displayed*
- [Rect](#) [d\\_contentRect](#)  
*holds content area so we can track changes.*
- float [d\\_vertStep](#)  
*vertical scroll step fraction.*
- float [d\\_vertOverlap](#)  
*vertical scroll overlap fraction.*
- float [d\\_horzStep](#)  
*horizontal scroll step fraction.*
- float [d\\_horzOverlap](#)  
*horizontal scroll overlap fraction.*
- [Event::Connection](#) [d\\_contentChangedConn](#)  
*[Event](#) connection to content pane.*
- [Event::Connection](#) [d\\_autoSizeChangedConn](#)  
*[Event](#) connection to content pane.*

### 6.236.1 Detailed Description

Base class for the [ScrollablePane](#) widget.

The [ScrollablePane](#) widget allows child windows to be attached which cover an area larger than the [ScrollablePane](#) itself and these child windows can be scrolled into view using the scrollbars of the scrollable pane.

## 6.236.2 Member Function Documentation

### 6.236.2.1 `const ScrolledContainer * CEGUI::ScrollablePane::getContentPane (void) const`

Returns a pointer to the window holding the pane contents.

The purpose of this is so that attached windows may be inspected, client code may not modify the returned window in any way.

**Returns:**

Pointer to the [ScrolledContainer](#) that is acting as the container for the scrollable pane content. The returned window is const, client code should not modify the [ScrolledContainer](#) settings directly.

### 6.236.2.2 `bool CEGUI::ScrollablePane::isVertScrollbarAlwaysShown (void) const`

Return whether the vertical scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

### 6.236.2.3 `void CEGUI::ScrollablePane::setShowVertScrollbar (bool setting)`

Set whether the vertical scroll bar should always be shown.

**Parameters:**

- setting*
- true if the vertical scroll bar should be shown even when it is not required.
  - false if the vertical scroll bar should only be shown when it is required.

**Returns:**

Nothing.

### 6.236.2.4 `bool CEGUI::ScrollablePane::isHorzScrollbarAlwaysShown (void) const`

Return whether the horizontal scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.236.2.5 void CEGUI::ScrollablePane::setShowHorzScrollbar (bool *setting*)**

Set whether the horizontal scroll bar should always be shown.

**Parameters:**

- setting*
- true if the horizontal scroll bar should be shown even when it is not required.
  - false if the horizontal scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.236.2.6 bool CEGUI::ScrollablePane::isContentPaneAutoSized (void) const**

Return whether the content pane is auto sized.

**Returns:**

- true to indicate the content pane will automatically resize itself.
- false to indicate the content pane will not automatically resize itself.

**6.236.2.7 void CEGUI::ScrollablePane::setContentPaneAutoSized (bool *setting*)**

Set whether the content pane should be auto-sized.

**Parameters:**

- setting*
- true to indicate the content pane should automatically resize itself.
  - false to indicate the content pane should not automatically resize itself.

**Returns:**

Nothing.

**6.236.2.8 const Rect & CEGUI::ScrollablePane::getContentPaneArea (void) const**

Return the current content pane area for the [ScrollablePane](#).

**Returns:**

[Rect](#) object that details the current pixel extents of the content pane attached to this [ScrollablePane](#).

**6.236.2.9 void CEGUI::ScrollablePane::setContentPaneArea (const Rect & *area*)**

Set the current content pane area for the [ScrollablePane](#).

**Note:**

If the [ScrollablePane](#) is configured to auto-size the content pane this call will have no effect.



**Parameters:**

*area* [Rect](#) object that details the pixel extents to use for the content pane attached to this [ScrollablePane](#).

**Returns:**

Nothing.

**6.236.2.10 float CEGUI::ScrollablePane::getHorizontalStepSize (void) const**

Returns the horizontal scrollbar step size as a fraction of one complete view page.

**Returns:**

float value specifying the step size where 1.0f would be the width of the viewing area.

**6.236.2.11 void CEGUI::ScrollablePane::setHorizontalStepSize (float *step*)**

Sets the horizontal scrollbar step size as a fraction of one complete view page.

**Parameters:**

*step* float value specifying the step size, where 1.0f would be the width of the viewing area.

**Returns:**

Nothing.

**6.236.2.12 float CEGUI::ScrollablePane::getHorizontalOverlapSize (void) const**

Returns the horizontal scrollbar overlap size as a fraction of one complete view page.

**Returns:**

float value specifying the overlap size where 1.0f would be the width of the viewing area.

**6.236.2.13 void CEGUI::ScrollablePane::setHorizontalOverlapSize (float *overlap*)**

Sets the horizontal scrollbar overlap size as a fraction of one complete view page.

**Parameters:**

*overlap* float value specifying the overlap size, where 1.0f would be the width of the viewing area.

**Returns:**

Nothing.

**6.236.2.14 float CEGUI::ScrollablePane::getHorizontalScrollPosition (void) const**

Returns the horizontal scroll position as a fraction of the complete scrollable width.

**Returns:**

float value specifying the scroll position.

**6.236.2.15 void CEGUI::ScrollablePane::setHorizontalScrollPosition (float *position*)**

Sets the horizontal scroll position as a fraction of the complete scrollable width.

**Parameters:**

*position* float value specifying the new scroll position.

**Returns:**

Nothing.

**6.236.2.16 float CEGUI::ScrollablePane::getVerticalStepSize (void) const**

Returns the vertical scrollbar step size as a fraction of one complete view page.

**Returns:**

float value specifying the step size where 1.0f would be the height of the viewing area.

**6.236.2.17 void CEGUI::ScrollablePane::setVerticalStepSize (float *step*)**

Sets the vertical scrollbar step size as a fraction of one complete view page.

**Parameters:**

*step* float value specifying the step size, where 1.0f would be the height of the viewing area.

**Returns:**

Nothing.

**6.236.2.18 float CEGUI::ScrollablePane::getVerticalOverlapSize (void) const**

Returns the vertical scrollbar overlap size as a fraction of one complete view page.

**Returns:**

float value specifying the overlap size where 1.0f would be the height of the viewing area.

**6.236.2.19 void CEGUI::ScrollablePane::setVerticalOverlapSize (float *overlap*)**

Sets the vertical scrollbar overlap size as a fraction of one complete view page.

**Parameters:**

*overlap* float value specifying the overlap size, where 1.0f would be the height of the viewing area.

**Returns:**

Nothing.

**6.236.2.20 float CEGUI::ScrollablePane::getVerticalScrollPosition (void) const**

Returns the vertical scroll position as a fraction of the complete scrollable height.

**Returns:**

float value specifying the scroll position.

**6.236.2.21 void CEGUI::ScrollablePane::setVerticalScrollPosition (float *position*)**

Sets the vertical scroll position as a fraction of the complete scrollable height.

**Parameters:**

*position* float value specifying the new scroll position.

**Returns:**

Nothing.

**6.236.2.22 Rect CEGUI::ScrollablePane::getViewableArea (void) const**

Return a [Rect](#) that described the pane's viewable area, relative to this [Window](#), in pixels.

**Returns:**

[Rect](#) object describing the ScrollablePane's viewable area.

**6.236.2.23 Scrollbar \* CEGUI::ScrollablePane::getVertScrollbar () const**

Return a pointer to the vertical scrollbar component widget for this [ScrollablePane](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the vertical [Scrollbar](#) component does not exist.

**6.236.2.24 Scrollbar \* CEGUI::ScrollablePane::getHorzScrollbar () const**

Return a pointer to the horizontal scrollbar component widget for this [ScrollablePane](#).

**Returns:**

Pointer to a [Scrollbar](#) object.

**Exceptions:**

*UnknownObjectException* Thrown if the horizontal [Scrollbar](#) component does not exist.

**6.236.2.25 void CEGUI::ScrollablePane::initialiseComponents (void) [virtual]**

Initialises the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Manager](#).

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.236.2.26 void CEGUI::ScrollablePane::destroy (void) [virtual]**

Internal destroy method which actually just adds the window and any parent destructed child windows to the dead pool.

This is virtual to allow for specialised cleanup which may be required in some advanced cases. If you override this for the above reason, you MUST call this base class version.

**Note:**

You never have to call this method yourself, use [WindowManager](#) to destroy your [Window](#) objects (which will call this for you).

Reimplemented from [CEGUI::Window](#).

**6.236.2.27 void CEGUI::ScrollablePane::configureScrollbars (void) [protected]**

Return a [Rect](#) that described the pane's viewable area, relative to this [Window](#), in pixels.

**Returns:**

[Rect](#) object describing the ScrollablePane's viewable area.

display required integrated scroll bars according to current size of the [ScrollablePane](#) view area and the size of the attached [ScrolledContainer](#).

**6.236.2.28 bool CEGUI::ScrollablePane::isVertScrollbarNeeded (void) const** [protected]

Return whether the vertical scrollbar is needed.

**Returns:**

- true if the scrollbar is either needed or forced via setting.
- false if the scrollbar should not be shown.

**6.236.2.29 bool CEGUI::ScrollablePane::isHorzScrollbarNeeded (void) const** [protected]

Return whether the horizontal scrollbar is needed.

**Returns:**

- true if the scrollbar is either needed or forced via setting.
- false if the scrollbar should not be shown.

**6.236.2.30 virtual bool CEGUI::ScrollablePane::testClassName\_impl (const String & class\_name) const** [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.236.2.31 ScrolledContainer \* CEGUI::ScrollablePane::getScrolledContainer () const** [protected]

Return a pointer to the [ScrolledContainer](#) component widget for this [ScrollablePane](#).

**Returns:**

Pointer to a [ScrolledContainer](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the scrolled container component does not exist.

**6.236.2.32** `virtual bool CEGUI::ScrollablePane::validateWindowRenderer (const String & name)  
const [inline, protected, virtual]`

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.236.2.33** `void CEGUI::ScrollablePane::onContentPaneChanged (WindowEventArgs & e)  
[protected, virtual]`

[Event](#) trigger method called when some pane content has changed size or location.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.236.2.34** `void CEGUI::ScrollablePane::onVertScrollbarModeChanged (WindowEventArgs & e)  
[protected, virtual]`

[Event](#) trigger method called when the setting that controls whether the vertical scrollbar is always shown or not, is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.236.2.35** `void CEGUI::ScrollablePane::onHorzScrollbarModeChanged (WindowEventArgs & e)  
[protected, virtual]`

[Event](#) trigger method called when the setting that controls whether the horizontal scrollbar is always shown or not, is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.236.2.36 void CEGUI::ScrollablePane::onAutoSizeSettingChanged (WindowEventArgs & *e*)**  
[protected, virtual]

Notification method called whenever the setting that controls whether the content pane is automatically sized is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.236.2.37 void CEGUI::ScrollablePane::onContentPaneScrolled (WindowEventArgs & *e*)**  
[protected, virtual]

Notification method called whenever the content pane is scrolled via changes in the scrollbar positions.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.236.2.38 void CEGUI::ScrollablePane::onSized (WindowEventArgs & *e*)** [protected, virtual]

Handler called when the window's size changes.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.236.2.39 void CEGUI::ScrollablePane::onMouseWheel (MouseEventArgs & *e*)** [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

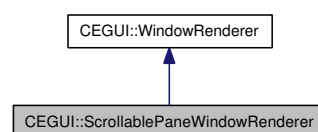
*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

## 6.237 CEGUI::ScrollablePaneWindowRenderer Class Reference

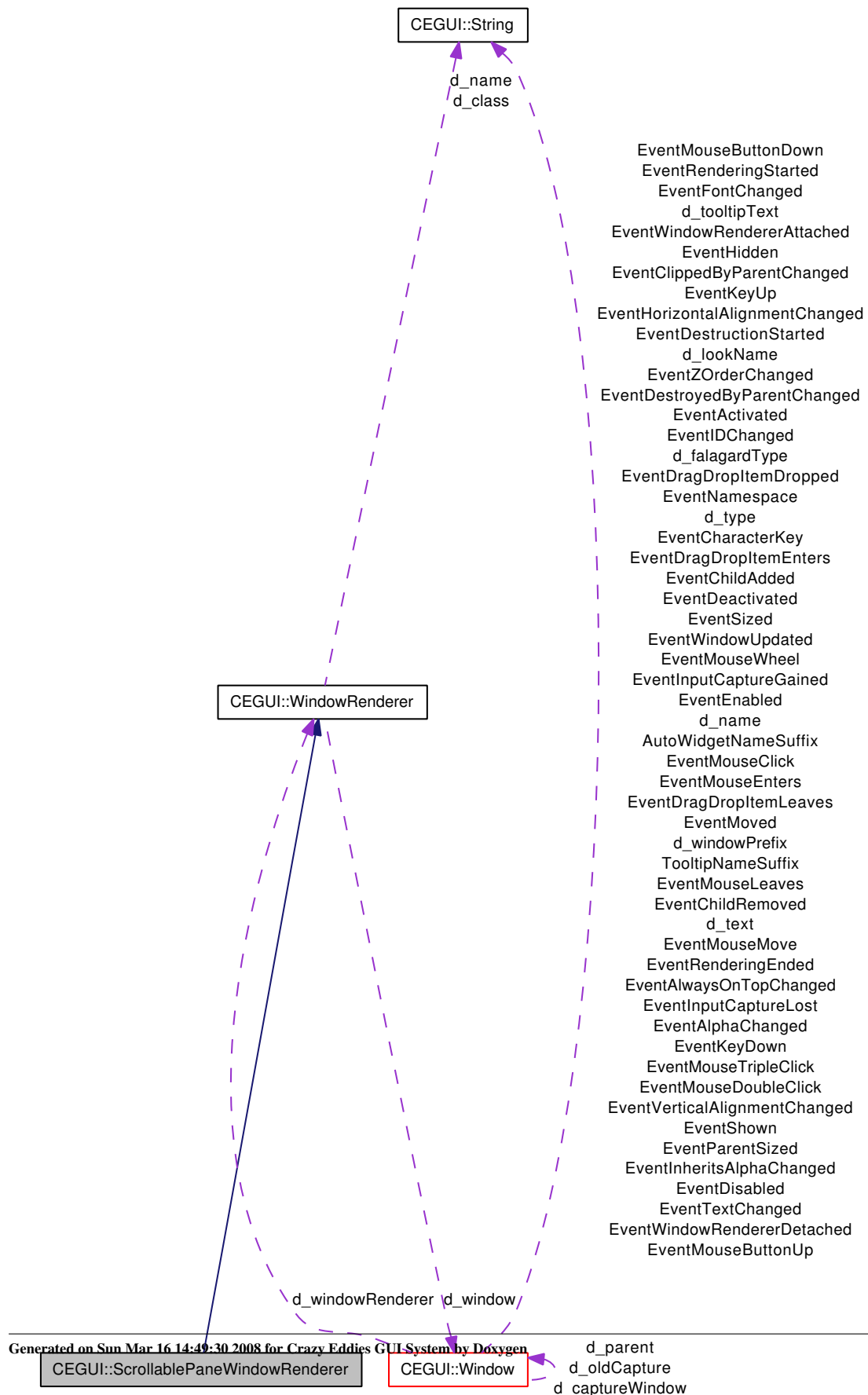
Base class for [ScrollablePane](#) window renderer objects.

Inheritance diagram for CEGUI::ScrollablePaneWindowRenderer:





Collaboration diagram for CEGUI::ScrollablePaneWindowRenderer:



## Public Member Functions

- [ScrollablePaneWindowRenderer](#) (const [String](#) &name)

*Constructor.*

- virtual [Rect](#) [getViewableArea](#) (void) const =0

*Return a [Rect](#) that described the pane's viewable area, relative to this [Window](#), in pixels.*

### 6.237.1 Detailed Description

Base class for [ScrollablePane](#) window renderer objects.

### 6.237.2 Member Function Documentation

#### 6.237.2.1 virtual [Rect](#) CEGUI::ScrollablePaneWindowRenderer::getViewableArea (void) const [pure virtual]

Return a [Rect](#) that described the pane's viewable area, relative to this [Window](#), in pixels.

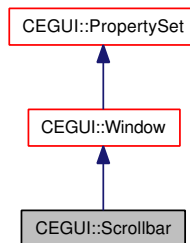
#### Returns:

[Rect](#) object describing the ScrollablePane's viewable area.

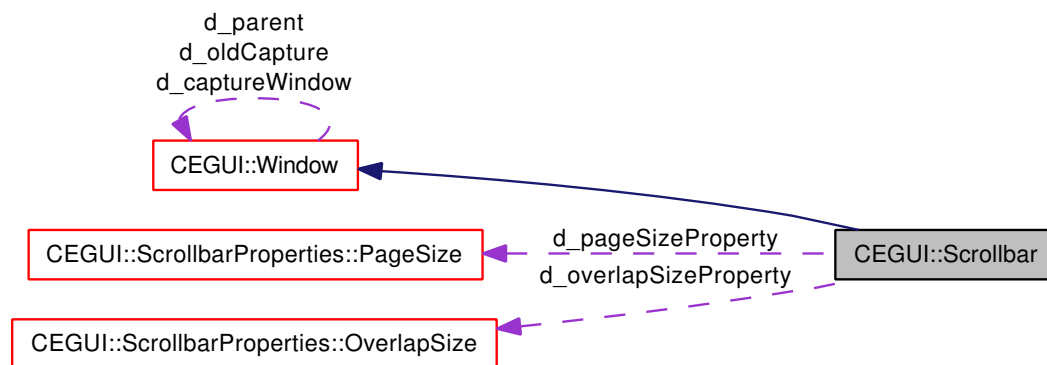
## 6.238 CEGUI::Scrollbar Class Reference

Base scroll bar class.

Inheritance diagram for CEGUI::Scrollbar:



Collaboration diagram for CEGUI::Scrollbar:



### Public Member Functions

- float [getDocumentSize](#) (void) const  
*Return the size of the document or data.*
- float [getPageSize](#) (void) const  
*Return the page size for this scroll bar.*
- float [getStepSize](#) (void) const  
*Return the step size for this scroll bar.*
- float [getOverlapSize](#) (void) const  
*Return the overlap size for this scroll bar.*
- float [getScrollPosition](#) (void) const  
*Return the current position of scroll bar within the document.*
- [PushButton](#) \* [getIncreaseButton](#) () const  
*Return a pointer to the 'increase' PushButtoncomponent widget for this [Scrollbar](#).*

- `PushButton * getDecreaseButton () const`  
*Return a pointer to the 'decrease' [PushButton](#) component widget for this [Scrollbar](#).*
- `Thumb * getThumb () const`  
*Return a pointer to the [Thumb](#) component widget for this [Scrollbar](#).*
- virtual void `initialiseComponents ()`  
*Initialises the [Scrollbar](#) object ready for use.*
- void `setDocumentSize (float document_size)`  
*Set the size of the document or data.*
- void `setPageSize (float page_size)`  
*Set the page size for this scroll bar.*
- void `setStepSize (float step_size)`  
*Set the step size for this scroll bar.*
- void `setOverlapSize (float overlap_size)`  
*Set the overlap size for this scroll bar.*
- void `setScrollPosition (float position)`  
*Set the current position of scroll bar within the document.*
- `Scrollbar (const String &type, const String &name)`  
*Constructor for [Scrollbar](#) objects.*
- virtual `~Scrollbar ()`  
*Destructor for [Scrollbar](#) objects.*

## Static Public Attributes

- static const `String EventNamespace`  
*Namespace for global events.*
- static const `String WidgetTypeName`  
*[Window](#) factory name.*
- static const `String EventScrollPositionChanged`  
*Name of the event fired when the scroll bar position value changes.*
- static const `String EventThumbTrackStarted`  
*Name of the event fired when the user begins dragging the thumb.*
- static const `String EventThumbTrackEnded`  
*Name of the event fired when the user releases the thumb.*

- static const [String EventScrollConfigChanged](#)  
*Name of the event fired when the scroll bar configuration data changes.*
- static const [String ThumbNameSuffix](#)  
*Widget name suffix for the thumb component.*
- static const [String IncreaseButtonNameSuffix](#)  
*Widget name suffix for the increase button component.*
- static const [String DecreaseButtonNameSuffix](#)  
*Widget name suffix for the decrease button component.*

## Protected Member Functions

- void [updateThumb](#) (void)  
*update the size and location of the thumb to properly represent the current state of the scroll bar*
- float [getValueFromThumb](#) (void) const  
*return value that best represents current scroll bar position given the current location of the thumb.*
- float [getAdjustDirectionFromPoint](#) (const [Point](#) &pt) const  
*Given window location pt, return a value indicating what change should be made to the scroll bar.*
- bool [handleThumbMoved](#) (const [EventArgs](#) &e)  
*update the size and location of the thumb to properly represent the current state of the scroll bar*
- bool [handleIncreaseClicked](#) (const [EventArgs](#) &e)  
*handler function for when the increase button is clicked.*
- bool [handleDecreaseClicked](#) (const [EventArgs](#) &e)  
*handler function for when the decrease button is clicked.*
- bool [handleThumbTrackStarted](#) (const [EventArgs](#) &e)  
*handler function for when thumb tracking begins*
- bool [handleThumbTrackEnded](#) (const [EventArgs](#) &e)  
*handler function for when thumb tracking begins*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- virtual void [onScrollPositionChanged](#) ([WindowEventArgs](#) &e)  
*Handler triggered when the scroll position changes.*
- virtual void [onThumbTrackStarted](#) ([WindowEventArgs](#) &e)

*Handler triggered when the user begins to drag the scroll bar thumb.*

- virtual void [onThumbTrackEnded](#) ([WindowEventArgs](#) &e)

*Handler triggered when the scroll bar thumb is released.*

- virtual void [onScrollConfigChanged](#) ([WindowEventArgs](#) &e)

*Handler triggered when the scroll bar data configuration changes.*

- virtual void [onMouseButtonDown](#) ([MouseEventArgs](#) &e)

*Handler called when a mouse button has been depressed within this window's area.*

- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)

*Handler called when the mouse wheel (z-axis) position changes within this window's area.*

## Protected Attributes

- float [d\\_documentSize](#)

*The size of the document / data being scrolled thorough.*

- float [d\\_pageSize](#)

*The size of a single 'page' of data.*

- float [d\\_stepSize](#)

*Step size used for increase / decrease button clicks.*

- float [d\\_overlapSize](#)

*Amount of overlap when jumping by a page.*

- float [d\\_position](#)

*Current scroll position.*

## 6.238.1 Detailed Description

Base scroll bar class.

This base class for scroll bars does not have any idea of direction - a derived class would add whatever meaning is appropriate according to what that derived class represents to the user.

## 6.238.2 Member Function Documentation

### 6.238.2.1 float CEGUI::Scrollbar::getDocumentSize (void) const [inline]

Return the size of the document or data.

The document size should be thought of as the total size of the data that is being scrolled through (the number of lines in a text file for example).

**Note:**

The returned value has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Returns:**

float value specifying the currently set document size.

**6.238.2.2 float CEGUI::Scrollbar::getPageSize (void) const [inline]**

Return the page size for this scroll bar.

The page size is typically the amount of data that can be displayed at one time. This value is also used when calculating the amount the position will change when you click either side of the scroll bar thumb - the amount the position changes will is (pageSize - overlapSize).

**Note:**

The returned value has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Returns:**

float value specifying the currently set page size.

**6.238.2.3 float CEGUI::Scrollbar::getStepSize (void) const [inline]**

Return the step size for this scroll bar.

The step size is typically a single unit of data that can be displayed, this is the amount the position will change when you click either of the arrow buttons on the scroll bar. (this could be 1 for a single line of text, for example).

**Note:**

The returned value has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Returns:**

float value specifying the currently set step size.

**6.238.2.4 float CEGUI::Scrollbar::getOverlapSize (void) const [inline]**

Return the overlap size for this scroll bar.

The overlap size is the amount of data from the end of a 'page' that will remain visible when the position is moved by a page. This is usually used so that the user keeps some context of where they were within the document's data when jumping a page at a time.

**Note:**

The returned value has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Returns:**

float value specifying the currently set overlap size.

**6.238.2.5 float CEGUI::Scrollbar::getScrollPosition (void) const** [inline]

Return the current position of scroll bar within the document.

The range of the scroll bar is from 0 to the size of the document minus the size of a page ( $0 \leq \text{position} \leq (\text{documentSize} - \text{pageSize})$ ).

**Note:**

The returned value has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Returns:**

float value specifying the current position of the scroll bar within its document.

**6.238.2.6 PushButton \* CEGUI::Scrollbar::getIncreaseButton () const**

Return a pointer to the 'increase' PushButtoncomponent widget for this [Scrollbar](#).

**Returns:**

Pointer to a [PushButton](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the increase [PushButton](#) component does not exist.

**6.238.2.7 PushButton \* CEGUI::Scrollbar::getDecreaseButton () const**

Return a pointer to the 'decrease' [PushButton](#) component widget for this [Scrollbar](#).

**Returns:**

Pointer to a [PushButton](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the 'decrease' [PushButton](#) component does not exist.

**6.238.2.8 Thumb \* CEGUI::Scrollbar::getThumb () const**

Return a pointer to the [Thumb](#) component widget for this [Scrollbar](#).

**Returns:**

Pointer to a [Thumb](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the [Thumb](#) component does not exist.



**6.238.2.9 void CEGUI::Scrollbar::initialiseComponents (void) [virtual]**

Initialises the [Scrollbar](#) object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.238.2.10 void CEGUI::Scrollbar::setDocumentSize (float *document\_size*)**

Set the size of the document or data.

The document size should be thought of as the total size of the data that is being scrolled through (the number of lines in a text file for example).

**Note:**

The value set has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Parameters:**

*document\_size* float value specifying the document size.

**Returns:**

Nothing.

**6.238.2.11 void CEGUI::Scrollbar::setPageSize (float *page\_size*)**

Set the page size for this scroll bar.

The page size is typically the amount of data that can be displayed at one time. This value is also used when calculating the amount the position will change when you click either side of the scroll bar thumb - the amount the position changes will is (pageSize - overlapSize).

**Note:**

The value set has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Parameters:**

*page\_size* float value specifying the page size.

**Returns:**

Nothing.

**6.238.2.12 void CEGUI::Scrollbar::setStepSize (float *step\_size*)**

Set the step size for this scroll bar.

The step size is typically a single unit of data that can be displayed, this is the amount the position will change when you click either of the arrow buttons on the scroll bar. (this could be 1 for a single line of text, for example).

**Note:**

The value set has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Parameters:**

*step\_size* float value specifying the step size.

**Returns:**

Nothing.

**6.238.2.13 void CEGUI::Scrollbar::setOverlapSize (float *overlap\_size*)**

Set the overlap size for this scroll bar.

The overlap size is the amount of data from the end of a 'page' that will remain visible when the position is moved by a page. This is usually used so that the user keeps some context of where they were within the document's data when jumping a page at a time.

**Note:**

The value set has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Parameters:**

*overlap\_size* float value specifying the overlap size.

**Returns:**

Nothing.

**6.238.2.14 void CEGUI::Scrollbar::setScrollPosition (float *position*)**

Set the current position of scroll bar within the document.

The range of the scroll bar is from 0 to the size of the document minus the size of a page ( $0 \leq \text{position} \leq (\text{documentSize} - \text{pageSize})$ ), any attempt to set the position outside this range will be adjusted so that it falls within the legal range.

**Note:**

The returned value has no meaning within the Gui system, it is left up to the application to assign appropriate values for the application specific use of the scroll bar.

**Parameters:**

*position* float value specifying the position of the scroll bar within its document.

**Returns:**

Nothing.

**6.238.2.15 float CEGUI::Scrollbar::getValueFromThumb (void) const** [protected]

return value that best represents current scroll bar position given the current location of the thumb.

**Returns:**

float value that, given the thumb widget position, best represents the current position for the scroll bar.

**6.238.2.16 float CEGUI::Scrollbar::getAdjustDirectionFromPoint (const Point & *pt*) const** [protected]

Given window location *pt*, return a value indicating what change should be made to the scroll bar.

**Parameters:**

*pt* Point object describing a pixel position in window space.

**Returns:**

- -1 to indicate scroll bar position should be moved to a lower value.
- 0 to indicate scroll bar position should not be changed.
- +1 to indicate scroll bar position should be moved to a higher value.

**6.238.2.17 bool CEGUI::Scrollbar::handleThumbMoved (const EventArgs & *e*)** [protected]

update the size and location of the thumb to properly represent the current state of the scroll bar

return value that best represents current scroll bar position given the current location of the thumb.

**Returns:**

float value that, given the thumb widget position, best represents the current position for the scroll bar.

Given window location *pt*, return a value indicating what change should be made to the scroll bar.

**Parameters:**

*pt* Point object describing a pixel position in window space.

**Returns:**

- -1 to indicate scroll bar position should be moved to a lower value.
- 0 to indicate scroll bar position should not be changed.
- +1 to indicate scroll bar position should be moved to a higher value.

handler function for when thumb moves.

**6.238.2.18** `virtual bool CEGUI::Scrollbar::testClassName_impl (const String & class_name) const`  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.238.2.19** `virtual bool CEGUI::Scrollbar::validateWindowRenderer (const String & name) const`  
[inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.238.2.20** `void CEGUI::Scrollbar::onMouseButtonDown (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.238.2.21** `void CEGUI::Scrollbar::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

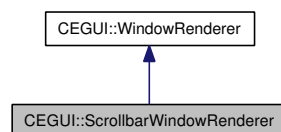
*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

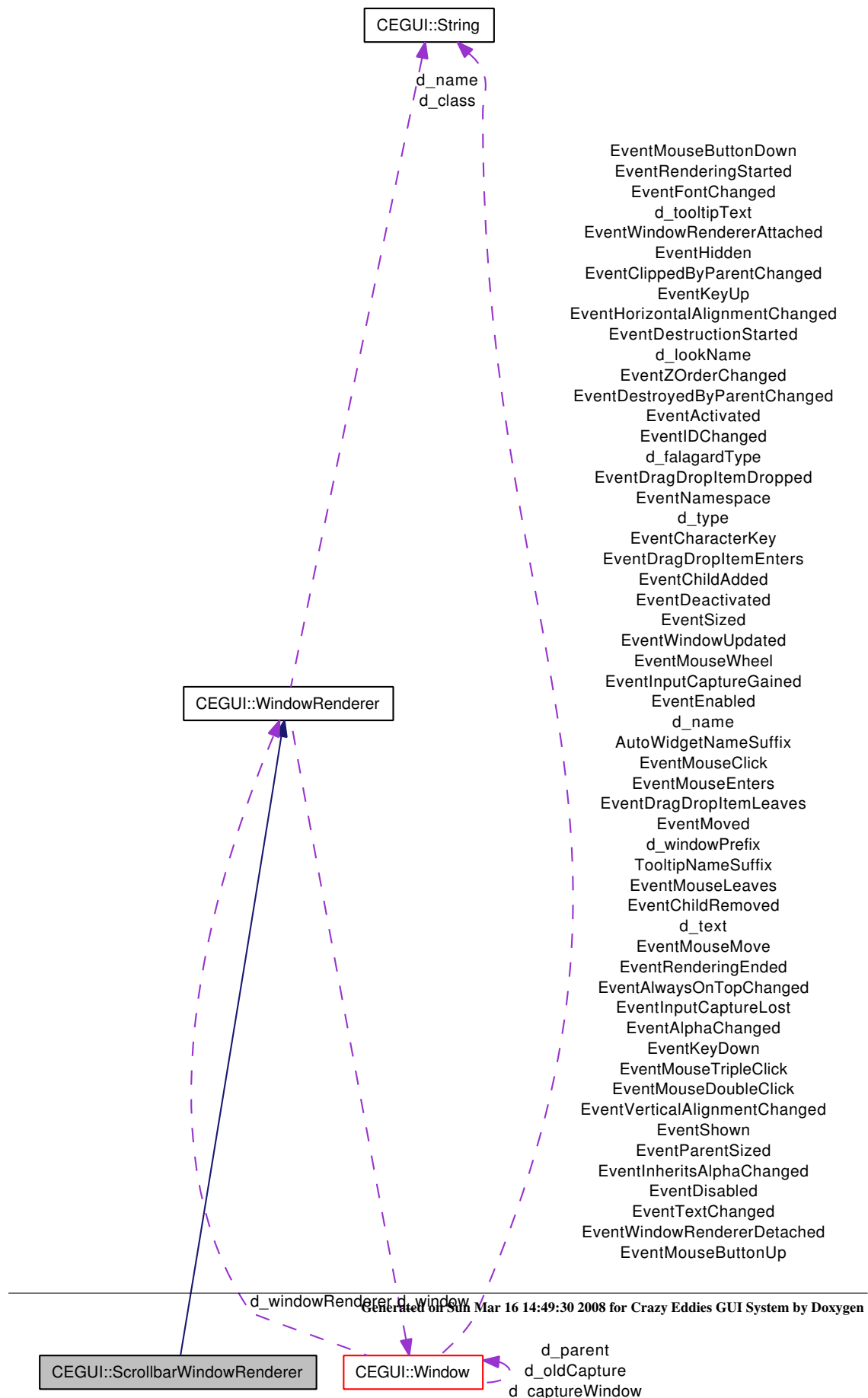
## 6.239 CEGUI::ScrollbarWindowRenderer Class Reference

Base class for [ItemEntry](#) window renderer objects.

Inheritance diagram for CEGUI::ScrollbarWindowRenderer:



Collaboration diagram for CEGUI::ScrollbarWindowRenderer:



## Public Member Functions

- [ScrollbarWindowRenderer](#) (const [String](#) &name)  
*Constructor.*
- virtual void [updateThumb](#) (void)=0  
*update the size and location of the thumb to properly represent the current state of the scroll bar*
- virtual float [getValueFromThumb](#) (void) const =0  
*return value that best represents current scroll bar position given the current location of the thumb.*
- virtual float [getAdjustDirectionFromPoint](#) (const [Point](#) &pt) const =0  
*Given window location pt, return a value indicating what change should be made to the scroll bar.*

### 6.239.1 Detailed Description

Base class for [ItemEntry](#) window renderer objects.

### 6.239.2 Member Function Documentation

#### 6.239.2.1 virtual float CEGUI::ScrollbarWindowRenderer::getValueFromThumb (void) const [pure virtual]

return value that best represents current scroll bar position given the current location of the thumb.

##### Returns:

float value that, given the thumb widget position, best represents the current position for the scroll bar.

#### 6.239.2.2 virtual float CEGUI::ScrollbarWindowRenderer::getAdjustDirectionFromPoint (const [Point](#) &pt) const [pure virtual]

Given window location *pt*, return a value indicating what change should be made to the scroll bar.

##### Parameters:

*pt* Point object describing a pixel position in window space.

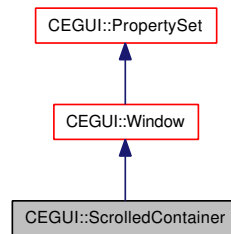
##### Returns:

- -1 to indicate scroll bar position should be moved to a lower value.
- 0 to indicate scroll bar position should not be changed.
- +1 to indicate scroll bar position should be moved to a higher value.

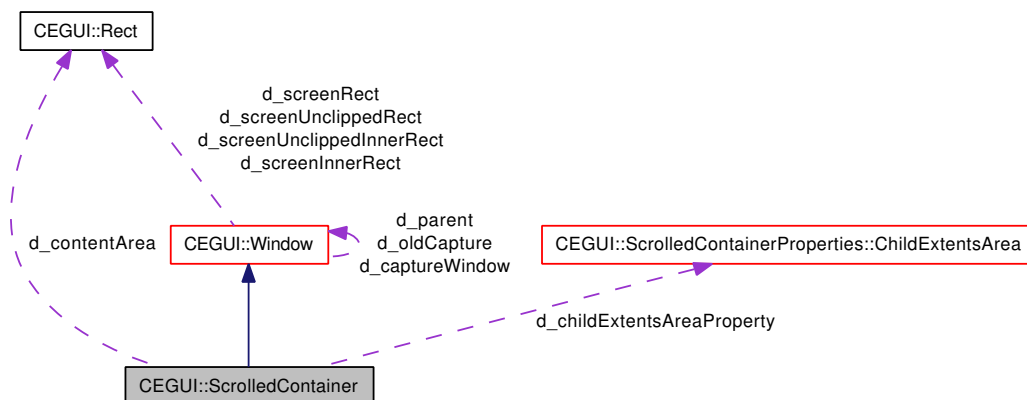
## 6.240 CEGUI::ScrolledContainer Class Reference

Helper container window class which is used in the implementation of the [ScrollablePane](#) widget class.

Inheritance diagram for CEGUI::ScrolledContainer:



Collaboration diagram for CEGUI::ScrolledContainer:



### Public Member Functions

- [ScrolledContainer](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [ScrolledContainer](#) objects.*
- [~ScrolledContainer](#) (void)  
*Destructor for [ScrolledContainer](#) objects.*
- bool [isContentPaneAutoSized](#) (void) const  
*Return whether the content pane is auto sized.*
- void [setContentPaneAutoSized](#) (bool setting)  
*Set whether the content pane should be auto-sized.*
- const [Rect](#) & [getContentArea](#) (void) const  
*Return the current content pane area for the [ScrolledContainer](#).*
- void [setContentArea](#) (const [Rect](#) &area)  
*Set the current content pane area for the [ScrolledContainer](#).*



- [Rect getChildExtentsArea](#) (void) const  
*Return the current extents of the attached content.*
- [Rect getUnclippedInnerRect\\_impl](#) (void) const  
*Return a [Rect](#) object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.*

## Static Public Attributes

- static const [String WidgetTypeName](#)  
*Type name for [ScrolledContainer](#).*
- static const [String EventNamespace](#)  
*Namespace for global events.*
- static const [String EventContentChanged](#)  
*[Event](#) fired whenever some child changes.*
- static const [String EventAutoSizeSettingChanged](#)  
*[Event](#) fired when the autosize setting changes.*

## Protected Types

- typedef std::multimap< [Window \\*](#), [Event::Connection](#) > [ConnectionTracker](#)

## Protected Member Functions

- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- void [drawSelf](#) (float z)  
*Perform the actual rendering for this [Window](#).*
- virtual void [onContentChanged](#) ([WindowEventArgs](#) &e)  
*Notification method called whenever the content size may have changed.*
- virtual void [onAutoSizeSettingChanged](#) ([WindowEventArgs](#) &e)  
*Notification method called whenever the setting that controls whether the content pane is automatically sized is changed.*
- bool [handleChildSized](#) (const [EventArgs](#) &e)  
*Method which receives notifications about child windows being moved.*
- bool [handleChildMoved](#) (const [EventArgs](#) &e)

*Method which receives notifications about child windows being sized.*

- void [onChildAdded](#) ([WindowEventArgs](#) &e)  
*Handler called when a child window is added to this window.*
- void [onChildRemoved](#) ([WindowEventArgs](#) &e)  
*Handler called when a child window is removed from this window.*
- void [onParentSized](#) ([WindowEventArgs](#) &e)  
*Handler called when this window's parent window has been resized. If this window is the root / GUI Sheet window, this call will be made when the display size changes.*

## Protected Attributes

- ConnectionTracker [d\\_eventConnections](#)  
*Tracks event connections we make.*
- [Rect](#) [d\\_contentArea](#)  
*Holds extents of the content pane.*
- bool [d\\_autosizePane](#)  
*true if the pane auto-sizes itself.*

## 6.240.1 Detailed Description

Helper container window class which is used in the implementation of the [ScrollablePane](#) widget class.

## 6.240.2 Member Function Documentation

### 6.240.2.1 bool CEGUI::ScrolledContainer::isContentPaneAutoSized (void) const

Return whether the content pane is auto sized.

#### Returns:

- true to indicate the content pane will automatically resize itself.
- false to indicate the content pane will not automatically resize itself.

### 6.240.2.2 void CEGUI::ScrolledContainer::setContentPaneAutoSized (bool *setting*)

Set whether the content pane should be auto-sized.

#### Parameters:

- setting* • true to indicate the content pane should automatically resize itself.
- false to indicate the content pane should not automatically resize itself.

**Returns:**

Nothing.

**6.240.2.3 const Rect & CEGUI::ScrolledContainer::getContentArea (void) const**

Return the current content pane area for the [ScrolledContainer](#).

**Returns:**

[Rect](#) object that details the current pixel extents of the content pane represented by this [ScrolledContainer](#).

**6.240.2.4 void CEGUI::ScrolledContainer::setContentArea (const Rect & area)**

Set the current content pane area for the [ScrolledContainer](#).

**Note:**

If the [ScrolledContainer](#) is configured to auto-size the content pane this call will have no effect.

**Parameters:**

*area* [Rect](#) object that details the pixel extents to use for the content pane represented by this [ScrolledContainer](#).

**Returns:**

Nothing.

**6.240.2.5 Rect CEGUI::ScrolledContainer::getChildExtentsArea (void) const**

Return the current extents of the attached content.

**Returns:**

[Rect](#) object that describes the pixel extents of the attached child windows. This is effectively the smallest bounding box that could contain all the attached windows.

**6.240.2.6 Rect CEGUI::ScrolledContainer::getUnclippedInnerRect\_impl (void) const**  
[virtual]

Return a [Rect](#) object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.

**Returns:**

[Rect](#) object that describes, in unclipped screen pixel co-ordinates, the window object's inner rect area.

Reimplemented from [CEGUI::Window](#).

**6.240.2.7** `virtual bool CEGUI::ScrolledContainer::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.240.2.8** `void CEGUI::ScrolledContainer::drawSelf (float z)` [inline, protected, virtual]

Perform the actual rendering for this [Window](#).

**Parameters:**

*z* float value specifying the base Z co-ordinate that should be used when rendering

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.240.2.9** `void CEGUI::ScrolledContainer::onContentChanged (WindowEventArgs & e)` [protected, virtual]

Notification method called whenever the content size may have changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.240.2.10** `void CEGUI::ScrolledContainer::onAutoSizeSettingChanged (WindowEventArgs & e)` [protected, virtual]

Notification method called whenever the setting that controls whether the content pane is automatically sized is changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.240.2.11 void CEGUI::ScrolledContainer::onChildAdded (WindowEventArgs & *e*)**  
[protected, virtual]

Handler called when a child window is added to this window.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that has been added.

Reimplemented from [CEGUI::Window](#).

**6.240.2.12 void CEGUI::ScrolledContainer::onChildRemoved (WindowEventArgs & *e*)**  
[protected, virtual]

Handler called when a child window is removed from this window.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set the window that has been removed.

Reimplemented from [CEGUI::Window](#).

**6.240.2.13 void CEGUI::ScrolledContainer::onParentSized (WindowEventArgs & *e*)**  
[protected, virtual]

Handler called when this window's parent window has been resized. If this window is the root / GUI Sheet window, this call will be made when the display size changes.

**Parameters:**

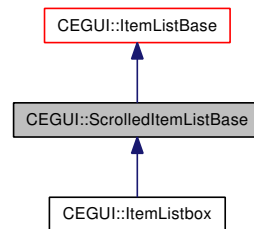
- e* [WindowEventArgs](#) object whose 'window' pointer field is set the the window that caused the event; this is typically either this window's parent window, or NULL to indicate the screen size has changed.

Reimplemented from [CEGUI::Window](#).

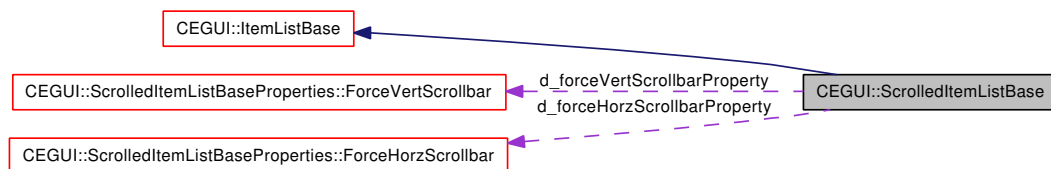
## 6.241 CEGUI::ScrolledItemListBase Class Reference

[ScrolledItemListBase](#) window class.

Inheritance diagram for CEGUI::ScrolledItemListBase:



Collaboration diagram for CEGUI::ScrolledItemListBase:



### Public Member Functions

- [bool isVertScrollbarAlwaysShown](#) (void) const  
*Returns whether the vertical scrollbar is being forced visible. Despite content size.*
- [bool isHorzScrollbarAlwaysShown](#) (void) const  
*Returns whether the horizontal scrollbar is being forced visible. Despite content size.*
- [Scrollbar \\* getVertScrollbar](#) () const  
*Get the vertical scrollbar component attached to this window.*
- [Scrollbar \\* getHorzScrollbar](#) () const  
*Get the horizontal scrollbar component attached to this window.*
- void [setShowVertScrollbar](#) (bool mode)  
*Sets whether the vertical scrollbar should be forced visible. Despite content size.*
- void [setShowHorzScrollbar](#) (bool mode)  
*Sets whether the horizontal scrollbar should be forced visible. Despite content size.*
- [ScrolledItemListBase](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for the [ScrolledItemListBase](#) base class constructor.*
- virtual [~ScrolledItemListBase](#) (void)  
*Destructor for the [ScrolledItemListBase](#) base class.*

- virtual void [initialiseComponents](#) (void)  
*Initialise the [Window](#) based object ready for use.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [VertScrollbarNameSuffix](#)  
*Name suffix for vertical scrollbar component.*
- static const [String](#) [HorzScrollbarNameSuffix](#)  
*Name suffix for horizontal scrollbar component.*
- static const [String](#) [ContentPaneNameSuffix](#)  
*Name suffix for the content pane component.*
- static const [String](#) [EventVertScrollbarModeChanged](#)  
*[Event](#) fired when the vertical scroll bar mode changes.*
- static const [String](#) [EventHorzScrollbarModeChanged](#)  
*[Event](#) fired when the horizontal scroll bar mode change.*

## Protected Member Functions

- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- void [configureScrollbars](#) (const [Size](#) &doc\_size)  
*Configure scrollbars.*
- virtual void [onVertScrollbarModeChanged](#) ([WindowEventArgs](#) &e)
- virtual void [onHorzScrollbarModeChanged](#) ([WindowEventArgs](#) &e)
- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*
- bool [handle\\_VScroll](#) (const [EventArgs](#) &e)
- bool [handle\\_HScroll](#) (const [EventArgs](#) &e)

## Protected Attributes

- bool [d\\_forceVScroll](#)
- bool [d\\_forceHScroll](#)

### 6.241.1 Detailed Description

[ScrolledItemListBox](#) window class.

### 6.241.2 Member Function Documentation

#### 6.241.2.1 void CEGUI::ScrolledItemListBox::initialiseComponents (void) [virtual]

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::ItemListBox](#).

#### 6.241.2.2 virtual bool CEGUI::ScrolledItemListBox::testClassName\_impl (const String & class\_name) const [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::ItemListBox](#).

Reimplemented in [CEGUI::ItemListbox](#).

#### 6.241.2.3 void CEGUI::ScrolledItemListBox::onMouseWheel (MouseEventArgs & e) [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

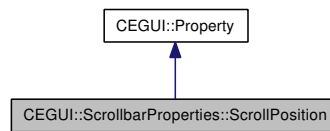
Reimplemented from [CEGUI::Window](#).



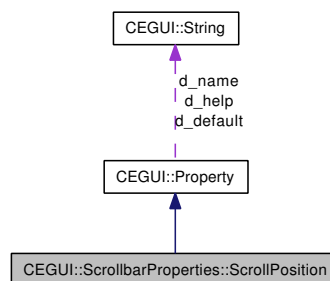
## 6.242 CEGUI::ScrollbarProperties::ScrollPosition Class Reference

[Property](#) to access the scroll position of the [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollbarProperties::ScrollPosition:



Collaboration diagram for CEGUI::ScrollbarProperties::ScrollPosition:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.242.1 Detailed Description

[Property](#) to access the scroll position of the [Scrollbar](#).

##### Usage:

- Name: [ScrollPosition](#)
- Format: "[float]".

##### Where:

- [float] specifies the current scroll position / value of the [Scrollbar](#).

## 6.242.2 Member Function Documentation

### 6.242.2.1 String CEGUI::ScrollbarProperties::ScrollPosition::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.242.2.2 void CEGUI::ScrollbarProperties::ScrollPosition::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

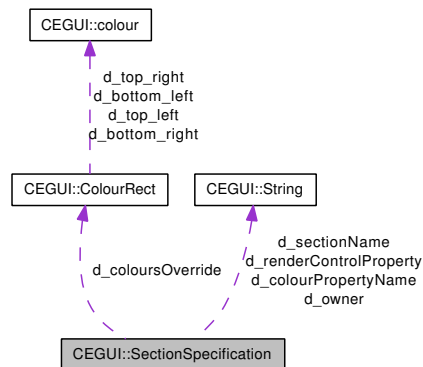
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.243 CEGUI::SectionSpecification Class Reference

Class that represents a simple 'link' to an [ImagerySection](#).

Collaboration diagram for CEGUI::SectionSpecification:



### Public Member Functions

- [SectionSpecification](#) (const [String](#) &owner, const [String](#) &sectionName, const [String](#) &controlPropertySource)

*Constructor.*

- [SectionSpecification](#) (const [String](#) &owner, const [String](#) &sectionName, const [String](#) &controlPropertySource, const [ColourRect](#) &cols)

*Constructor.*

- void [render](#) ([Window](#) &srcWindow, float base\_z, const [ColourRect](#) \*modcols=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const

*Render the section specified by this [SectionSpecification](#).*

- void [render](#) ([Window](#) &srcWindow, const [Rect](#) &baseRect, float base\_z, const [ColourRect](#) \*modcols=0, const [Rect](#) \*clipper=0, bool clipToDisplay=false) const

*Render the section specified by this [SectionSpecification](#).*

- const [String](#) & [getOwnerWidgetLookFeel](#) () const

*Return the name of the [WidgetLookFeel](#) object containing the target section.*

- const [String](#) & [getSectionName](#) () const

*Return the name of the target [ImagerySection](#).*

- const [ColourRect](#) & [getOverrideColours](#) () const

*Return the current override colours.*

- void [setOverrideColours](#) (const [ColourRect](#) &cols)

*Set the override colours to be used by this [SectionSpecification](#).*

- bool [isUsingOverrideColours](#) () const

return whether the use of override colours is enabled on this [SectionSpecification](#).

- void [setUsingOverrideColours](#) (bool setting=true)  
*Enable or disable the use of override colours for this section.*
- void [setOverrideColoursPropertySource](#) (const [String](#) &property)  
*Set the name of the property where override [colour](#) values can be obtained.*
- void [setOverrideColoursPropertyIsColourRect](#) (bool setting=true)  
*Set whether the override colours property source represents a full [ColourRect](#).*
- void [setRenderControlPropertySource](#) (const [String](#) &property)  
*Set the name of the property that controls whether to actually render this section.*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [SectionSpecification](#) to out\_stream.*

## Protected Member Functions

- void [initColourRectForOverride](#) (const [Window](#) &wnd, [ColourRect](#) &cr) const  
*Helper method to initialise a [ColourRect](#) with appropriate values according to the way the section specification is set up.*

### 6.243.1 Detailed Description

Class that represents a simple 'link' to an [ImagerySection](#).

This class enables sections to be easily re-used, by different states and/or layers, by allowing sections to be specified by name rather than having multiple copies of the same thing all over the place.

### 6.243.2 Constructor & Destructor Documentation

#### 6.243.2.1 CEGUI::SectionSpecification::SectionSpecification (const [String](#) & owner, const [String](#) & sectionName, const [String](#) & controlPropertySource)

Constructor.

#### Parameters:

*owner* [String](#) holding the name of the [WidgetLookAndFeel](#) object that contains the target section.

*sectionName* [String](#) holding the name of the target section.

*controlPropertySource* [String](#) holding the name of a boolean property that will control if the rendering for this section will actually occur or not.

### 6.243.2.2 CEGUI::SectionSpecification::SectionSpecification (const String & owner, const String & sectionName, const String & controlPropertySource, const ColourRect & cols)

Constructor.

#### Parameters:

- owner* [String](#) holding the name of the [WidgetLookAndFeel](#) object that contains the target section.
- sectionName* [String](#) holding the name of the target section.
- controlPropertySource* [String](#) holding the name of a boolean property that will control if the rendering for this section will actually occur or not.
- cols* Override colours to be used (modulates sections master colours).

### 6.243.3 Member Function Documentation

#### 6.243.3.1 void CEGUI::SectionSpecification::render (Window & srcWindow, float base\_z, const ColourRect \* modcols = 0, const Rect \* clipper = 0, bool clipToDisplay = false) const

Render the section specified by this [SectionSpecification](#).

#### Parameters:

- srcWindow* [Window](#) object to be used when calculating pixel values from [BaseDim](#) values.
- base\_z* base z co-ordinate to use for all imagery in the linked section.

#### Returns:

Nothing.

#### 6.243.3.2 void CEGUI::SectionSpecification::render (Window & srcWindow, const Rect & baseRect, float base\_z, const ColourRect \* modcols = 0, const Rect \* clipper = 0, bool clipToDisplay = false) const

Render the section specified by this [SectionSpecification](#).

#### Parameters:

- srcWindow* [Window](#) object to be used when calculating pixel values from [BaseDim](#) values.
- baseRect* [Rect](#) object to be used when calculating pixel values from [BaseDim](#) values.
- base\_z* base z co-ordinate to use for all imagery in the linked section.

#### Returns:

Nothing.

#### 6.243.3.3 const String & CEGUI::SectionSpecification::getOwnerWidgetLookAndFeel () const

Return the name of the [WidgetLookAndFeel](#) object containing the target section.

#### Returns:

- [String](#) object holding the name of the [WidgetLookAndFeel](#) that contains the target [ImagerySection](#).

**6.243.3.4 const String & CEGUI::SectionSpecification::getSectionName () const**

Return the name of the target [ImagerySection](#).

**Returns:**

[String](#) object holding the name of the target [ImagerySection](#).

**6.243.3.5 const ColourRect & CEGUI::SectionSpecification::getOverrideColours () const**

Return the current override colours.

**Returns:**

[ColourRect](#) holding the colours that will be modulated with the sections master colours if [colour](#) override is enabled on this [SectionSpecification](#).

**6.243.3.6 void CEGUI::SectionSpecification::setOverrideColours (const ColourRect & cols)**

Set the override colours to be used by this [SectionSpecification](#).

**Parameters:**

*cols* [ColourRect](#) describing the override colours to set for this [SectionSpecification](#).

**Returns:**

Nothing.

**6.243.3.7 bool CEGUI::SectionSpecification::isUsingOverrideColours () const**

return whether the use of override colours is enabled on this [SectionSpecification](#).

**Returns:**

- true if override colours will be used for this [SectionSpecification](#).
- false if override colours will not be used for this [SectionSpecification](#).

**6.243.3.8 void CEGUI::SectionSpecification::setUsingOverrideColours (bool setting = true)**

Enable or disable the use of override colours for this section.

**Parameters:**

- setting*
- true if override colours should be used for this [SectionSpecification](#).
  - false if override colours should not be used for this [SectionSpecification](#).

**Returns:**

Nothing.

**6.243.3.9 void CEGUI::SectionSpecification::setOverrideColoursPropertySource (const String & *property*)**

Set the name of the property where override [colour](#) values can be obtained.

**Parameters:**

*property* [String](#) containing the name of the property.

**Returns:**

Nothing.

**6.243.3.10 void CEGUI::SectionSpecification::setOverrideColoursPropertyIsColourRect (bool *setting* = true)**

Set whether the override colours property source represents a full [ColourRect](#).

**Parameters:**

- setting* • true if the override colours property will access a [ColourRect](#) object.
- false if the override colours property will access a [colour](#) object.

**Returns:**

Nothing.

**6.243.3.11 void CEGUI::SectionSpecification::setRenderControlPropertySource (const String & *property*)**

Set the name of the property that controls whether to actually render this section.

**Parameters:**

*property* [String](#) containing the name of the property.

**Returns:**

Nothing.

**6.243.3.12 void CEGUI::SectionSpecification::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [SectionSpecification](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.243.3.13 void CEGUI::SectionSpecification::initColourRectForOverride (const Window & *wnd*, ColourRect & *cr*) const** [protected]

Helper method to initialise a [ColourRect](#) with appropriate values according to the way the section specification is set up.

This will try and get values from multiple places:

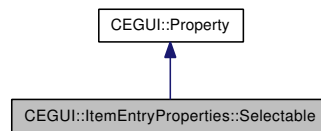
- a property attached to *wnd*
- the integral `d_coloursOverride` values.
- or default to `colour(1,1,1,1)`;



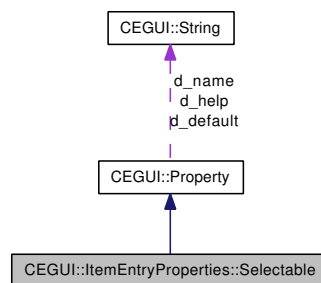
## 6.244 CEGUI::ItemEntryProperties::Selectable Class Reference

[Property](#) to access the state of the selectable setting.

Inheritance diagram for CEGUI::ItemEntryProperties::Selectable:



Collaboration diagram for CEGUI::ItemEntryProperties::Selectable:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.244.1 Detailed Description

[Property](#) to access the state of the selectable setting.

**Usage:**

- Name: [Selectable](#)
- Format: "[text]".

**Where [text] is:**

- "True" to indicate that the item is selectable.
- "False" to indicate that the item is not selectable.

## 6.244.2 Member Function Documentation

### 6.244.2.1 `String CEGUI::ItemEntryProperties::Selectable::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.244.2.2 `void CEGUI::ItemEntryProperties::Selectable::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

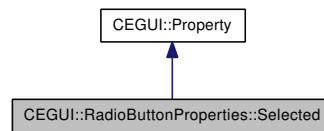
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

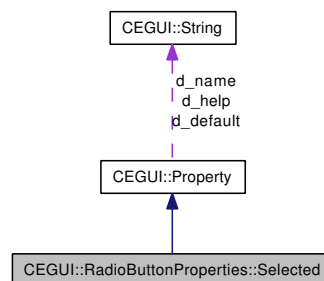
## 6.245 CEGUI::RadioButtonProperties::Selected Class Reference

[Property](#) to access the selected state of the [RadioButton](#).

Inheritance diagram for CEGUI::RadioButtonProperties::Selected:



Collaboration diagram for CEGUI::RadioButtonProperties::Selected:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.245.1 Detailed Description

[Property](#) to access the selected state of the [RadioButton](#).

**Usage:**

- Name: [Selected](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the radio button is selected.
- "False" to indicate the radio button is not selected.

## 6.245.2 Member Function Documentation

### 6.245.2.1 `String CEGUI::RadioButtonProperties::Selected::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.245.2.2 `void CEGUI::RadioButtonProperties::Selected::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

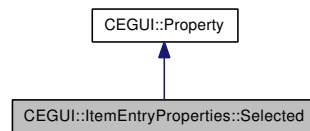
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

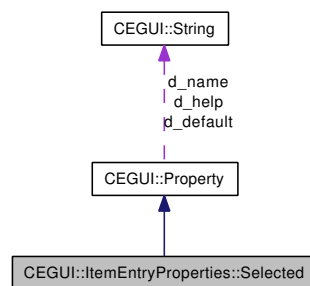
## 6.246 CEGUI::ItemEntryProperties::Selected Class Reference

[Property](#) to access the state of the selected setting.

Inheritance diagram for CEGUI::ItemEntryProperties::Selected:



Collaboration diagram for CEGUI::ItemEntryProperties::Selected:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.246.1 Detailed Description

[Property](#) to access the state of the selected setting.

**Usage:**

- Name: [Selected](#)
- Format: "[text]".

**Where [text] is:**

- "True" to indicate that the item is selectable.
- "False" to indicate that the item is not selectable.

## 6.246.2 Member Function Documentation

### 6.246.2.1 `String CEGUI::ItemEntryProperties::Selected::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.246.2.2 `void CEGUI::ItemEntryProperties::Selected::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

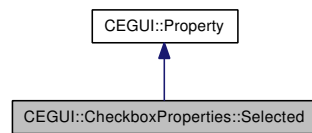
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

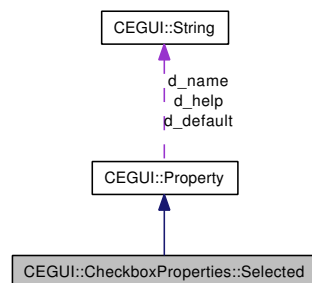
## 6.247 CEGUI::CheckboxProperties::Selected Class Reference

[Property](#) to access the selected state of the check box.

Inheritance diagram for CEGUI::CheckboxProperties::Selected:



Collaboration diagram for CEGUI::CheckboxProperties::Selected:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.247.1 Detailed Description

[Property](#) to access the selected state of the check box.

This property offers access to the select state for the [Checkbox](#) object.

Usage:

- Name: [Selected](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the check box is selected (has check mark).
- "False" to indicate the check box is not selected (does not have check mark).

## 6.247.2 Member Function Documentation

### 6.247.2.1 `String CEGUI::CheckboxProperties::Selected::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.247.2.2 `void CEGUI::CheckboxProperties::Selected::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

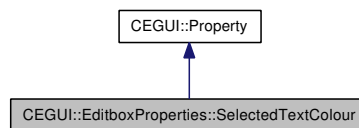
Implements [CEGUI::Property](#).



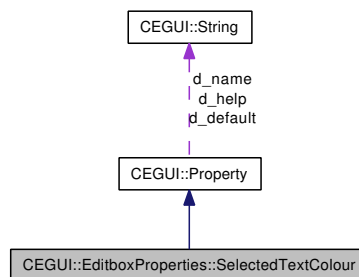
## 6.248 CEGUI::EditboxProperties::SelectedTextColour Class Reference

[Property](#) to access the [colour](#) used for rendering text within the selection area.

Inheritance diagram for CEGUI::EditboxProperties::SelectedTextColour:



Collaboration diagram for CEGUI::EditboxProperties::SelectedTextColour:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.248.1 Detailed Description

[Property](#) to access the [colour](#) used for rendering text within the selection area.

##### Usage:

- Name: [SelectedTextColour](#)
- Format: "aarrggbb".

##### Where:

- aarrggbb is the ARGB [colour](#) value to be used.

## 6.248.2 Member Function Documentation

### 6.248.2.1 String CEGUI::EditboxProperties::SelectedTextColour::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.248.2.2 void CEGUI::EditboxProperties::SelectedTextColour::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

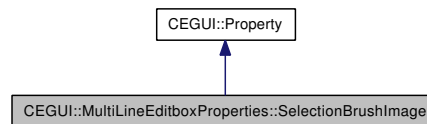
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

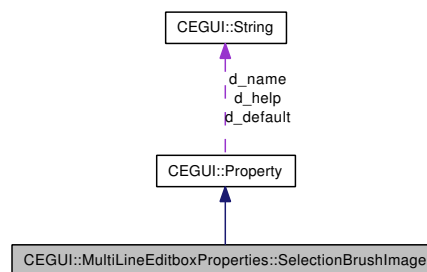
## 6.249 CEGUI::MultiLineEditboxProperties::SelectionBrushImage Class Reference

[Property](#) to access the selection brush image.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::SelectionBrushImage:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::SelectionBrushImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.249.1 Detailed Description

[Property](#) to access the selection brush image.

Usage:

- Name: [SelectionBrushImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.249.2 Member Function Documentation

#### 6.249.2.1 String CEGUI::MultiLineEditboxProperties::SelectionBrushImage::get (const [PropertyReceiver](#) \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.249.2.2** `void CEGUI::MultiLineEditboxProperties::SelectionBrushImage::set  
(PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

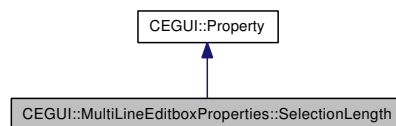
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

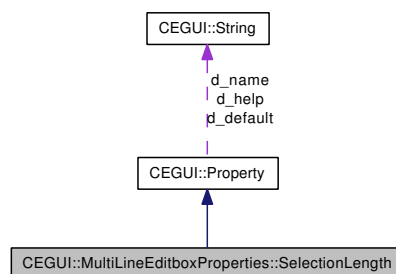
## 6.250 CEGUI::MultiLineEditboxProperties::SelectionLength Class Reference

[Property](#) to access the current selection length.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::SelectionLength:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::SelectionLength:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.250.1 Detailed Description

[Property](#) to access the current selection length.

#### Usage:

- Name: [SelectionLength](#)
- Format: "[uint]"

#### Where:

- [uint] is the length of the selection (as a count of the number of code points selected).

## 6.250.2 Member Function Documentation

### 6.250.2.1 String CEGUI::MultiLineEditboxProperties::SelectionLength::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.250.2.2 void CEGUI::MultiLineEditboxProperties::SelectionLength::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

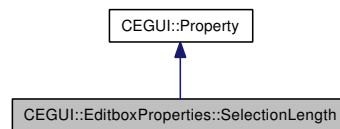
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

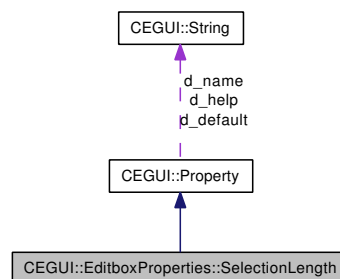
## 6.251 CEGUI::EditboxProperties::SelectionLength Class Reference

[Property](#) to access the current selection length.

Inheritance diagram for CEGUI::EditboxProperties::SelectionLength:



Collaboration diagram for CEGUI::EditboxProperties::SelectionLength:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.251.1 Detailed Description

[Property](#) to access the current selection length.

**Usage:**

- Name: [SelectionLength](#)
- Format: "[uint]"

**Where:**

- [uint] is the length of the selection (as a count of the number of code points selected).

## 6.251.2 Member Function Documentation

### 6.251.2.1 `String CEGUI::EditboxProperties::SelectionLength::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.251.2.2 `void CEGUI::EditboxProperties::SelectionLength::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

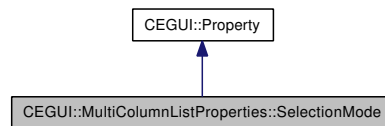
Implements [CEGUI::Property](#).



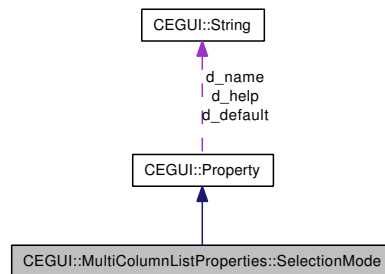
## 6.252 CEGUI::MultiColumnListProperties::SelectionMode Class Reference

[Property](#) to access the selection mode setting of the list.

Inheritance diagram for CEGUI::MultiColumnListProperties::SelectionMode:



Collaboration diagram for CEGUI::MultiColumnListProperties::SelectionMode:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.252.1 Detailed Description

[Property](#) to access the selection mode setting of the list.

**Usage:**

- Name: [SelectionMode](#)
- Format: "[text]"

Where [Text] is one of:

- "RowSingle"
- "RowMultiple"
- "CellSingle"
- "CellMultiple"

- "NominatedColumnSingle"
- "NominatedColumnMultiple"
- "ColumnSingle"
- "ColumnMultiple"
- "NominatedRowSingle"
- "NominatedRowMultiple"

## 6.252.2 Member Function Documentation

### 6.252.2.1 String CEGUI::MultiColumnListProperties::SelectionMode::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.252.2.2 void CEGUI::MultiColumnListProperties::SelectionMode::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

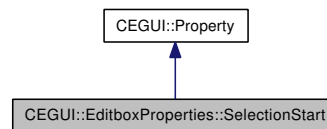
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

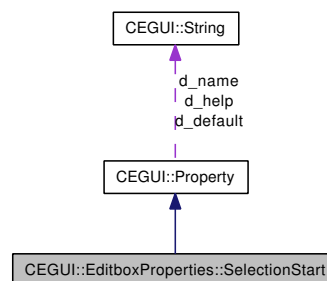
## 6.253 CEGUI::EditboxProperties::SelectionStart Class Reference

[Property](#) to access the current selection start index.

Inheritance diagram for CEGUI::EditboxProperties::SelectionStart:



Collaboration diagram for CEGUI::EditboxProperties::SelectionStart:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.253.1 Detailed Description

[Property](#) to access the current selection start index.

**Usage:**

- Name: [SelectionStart](#)
- Format: "[uint]"

**Where:**

- [uint] is the zero based index of the selection start position within the text.

## 6.253.2 Member Function Documentation

### 6.253.2.1 `String CEGUI::EditboxProperties::SelectionStart::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.253.2.2 `void CEGUI::EditboxProperties::SelectionStart::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

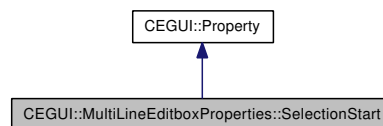
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

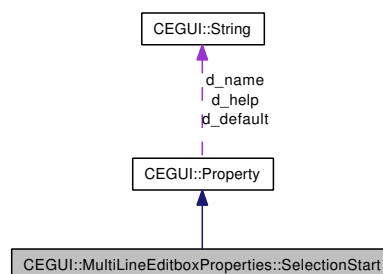
## 6.254 CEGUI::MultiLineEditboxProperties::SelectionStart Class Reference

[Property](#) to access the current selection start index.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::SelectionStart:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::SelectionStart:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.254.1 Detailed Description

[Property](#) to access the current selection start index.

##### Usage:

- Name: [SelectionStart](#)
- Format: "[uint]"

##### Where:

- [uint] is the zero based index of the selection start position within the text.

## 6.254.2 Member Function Documentation

### 6.254.2.1 String CEGUI::MultiLineEditboxProperties::SelectionStart::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.254.2.2 void CEGUI::MultiLineEditboxProperties::SelectionStart::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.255 CEGUI::SimpleTimer Class Reference

Simple timer class.

### Public Member Functions

- void **restart** ()
- double **elapsed** ()

### Static Public Member Functions

- static double **currentTime** ()  
*returns time in seconds*

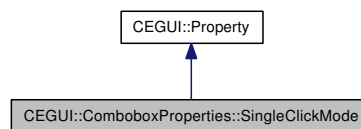
#### 6.255.1 Detailed Description

Simple timer class.

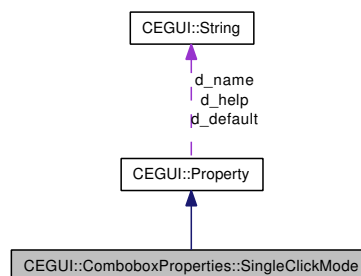
## 6.256 CEGUI::ComboboxProperties::SingleClickMode Class Reference

[Property](#) to access the 'single click mode' setting for the combo box.

Inheritance diagram for CEGUI::ComboboxProperties::SingleClickMode:



Collaboration diagram for CEGUI::ComboboxProperties::SingleClickMode:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.256.1 Detailed Description

[Property](#) to access the 'single click mode' setting for the combo box.

**Usage:**

- Name: [SingleClickMode](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate that the box will operate in single click mode
- "False" to indicate that the box will not operate in single click mode



## 6.256.2 Member Function Documentation

### 6.256.2.1 String CEGUI::ComboboxProperties::SingleClickMode::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.256.2.2 void CEGUI::ComboboxProperties::SingleClickMode::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

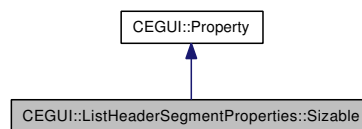
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

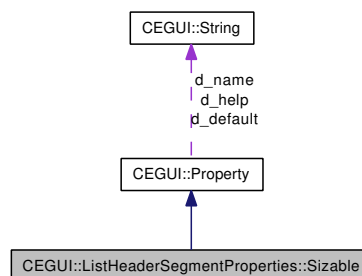
## 6.257 CEGUI::ListHeaderSegmentProperties::Sizable Class Reference

[Property](#) to access the sizable setting of the header segment.

Inheritance diagram for CEGUI::ListHeaderSegmentProperties::Sizable:



Collaboration diagram for CEGUI::ListHeaderSegmentProperties::Sizable:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.257.1 Detailed Description

[Property](#) to access the sizable setting of the header segment.

Usage:

- Name: [Sizable](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the segment can be sized by the user.
- "False" to indicate the segment can not be sized by the user.

## 6.257.2 Member Function Documentation

### 6.257.2.1 String CEGUI::ListHeaderSegmentProperties::Sizable::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.257.2.2 void CEGUI::ListHeaderSegmentProperties::Sizable::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.258 CEGUI::Size Class Reference

Class that holds the size (width & height) of something.

### Public Member Functions

- **Size** (float width, float height)
- bool **operator==** (const [Size](#) &other) const
- bool **operator!=** (const [Size](#) &other) const

### Public Attributes

- float **d\_width**
- float **d\_height**

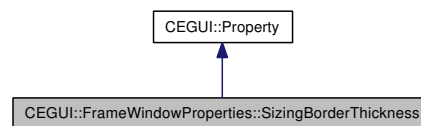
#### 6.258.1 Detailed Description

Class that holds the size (width & height) of something.

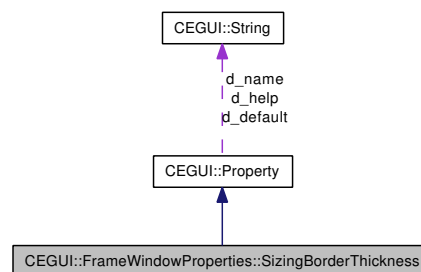
## 6.259 CEGUI::FrameWindowProperties::SizingBorderThickness Class Reference

[Property](#) to access the setting for the sizing border thickness.

Inheritance diagram for CEGUI::FrameWindowProperties::SizingBorderThickness:



Collaboration diagram for CEGUI::FrameWindowProperties::SizingBorderThickness:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.259.1 Detailed Description

[Property](#) to access the setting for the sizing border thickness.

#### Usage:

- Name: [SizingBorderThickness](#)
- Format: "[float]".

#### Where:

- [float] is the size of the invisible sizing border in screen pixels.

## 6.259.2 Member Function Documentation

### 6.259.2.1 String CEGUI::FrameWindowProperties::SizingBorderThickness::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.259.2.2 void CEGUI::FrameWindowProperties::SizingBorderThickness::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

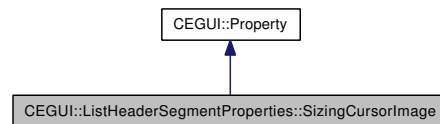
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

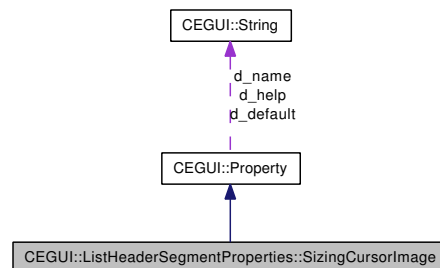
## 6.260 CEGUI::ListHeaderSegmentProperties::SizingCursorImage Class Reference

[Property](#) to access the segment sizing cursor image.

Inheritance diagram for CEGUI::ListHeaderSegmentProperties::SizingCursorImage:



Collaboration diagram for CEGUI::ListHeaderSegmentProperties::SizingCursorImage:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.260.1 Detailed Description

[Property](#) to access the segment sizing cursor image.

Usage:

- Name: [SizingCursorImage](#)
- Format: "set:<imageset> image:<imagename>".

### 6.260.2 Member Function Documentation

#### 6.260.2.1 String CEGUI::ListHeaderSegmentProperties::SizingCursorImage::get (const [PropertyReceiver](#) \* receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.260.2.2** `void CEGUI::ListHeaderSegmentProperties::SizingCursorImage::set  
(PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[\*InvalidRequestException\*](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

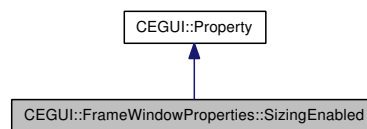
Implements [CEGUI::Property](#).



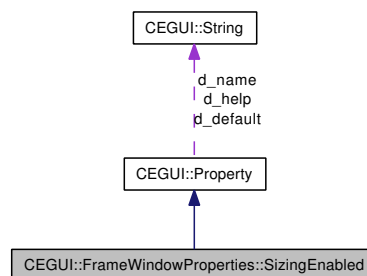
## 6.261 CEGUI::FrameWindowProperties::SizingEnabled Class Reference

[Property](#) to access the state of the sizable setting for the [FrameWindow](#).

Inheritance diagram for CEGUI::FrameWindowProperties::SizingEnabled:



Collaboration diagram for CEGUI::FrameWindowProperties::SizingEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.261.1 Detailed Description

[Property](#) to access the state of the sizable setting for the [FrameWindow](#).

**Usage:**

- Name: [SizingEnabled](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the window will be user re-sizable.
- "False" to indicate the window will not be re-sizable by the user.

## 6.261.2 Member Function Documentation

### 6.261.2.1 String CEGUI::FrameWindowProperties::SizingEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.261.2.2 void CEGUI::FrameWindowProperties::SizingEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

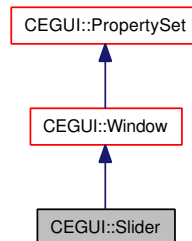
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

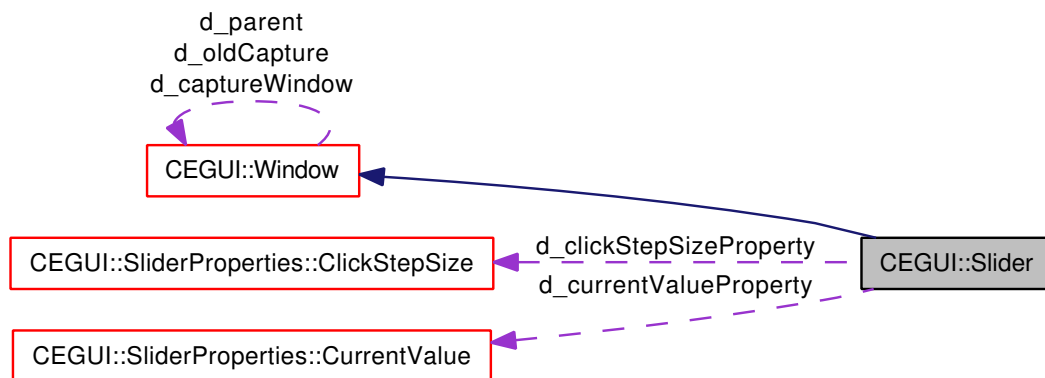
## 6.262 CEGUI::Slider Class Reference

Base class for [Slider](#) widgets.

Inheritance diagram for CEGUI::Slider:



Collaboration diagram for CEGUI::Slider:



### Public Member Functions

- float [getCurrentValue](#) (void) const  
*return the current slider value.*
- float [getMaxValue](#) (void) const  
*return the maximum value set for this widget*
- float [getClickStep](#) (void) const  
*return the current click step setting for the slider.*
- [Thumb](#) \* [getThumb](#) () const  
*Return a pointer to the [Thumb](#) component widget for this [Slider](#).*
- virtual void [initialiseComponents](#) (void)  
*Initialises the [Window](#) based object ready for use.*
- void [setMaxValue](#) (float maxVal)  
*set the maximum value for the slider. Note that the minimum value is fixed at 0.*

- void [setCurrentValue](#) (float value)  
*set the current slider value.*
- void [setClickStep](#) (float step)  
*set the current click step setting for the slider.*
- [Slider](#) (const [String](#) &type, const [String](#) &name)  
*[Slider](#) base class constructor.*
- virtual [~Slider](#) (void)  
*[Slider](#) base class destructor.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventValueChanged](#)  
*[Event](#) fired when the slider value changes.*
- static const [String](#) [EventThumbTrackStarted](#)  
*Name of the event fired when the user begins dragging the thumb.*
- static const [String](#) [EventThumbTrackEnded](#)  
*Name of the event fired when the user releases the thumb.*
- static const [String](#) [ThumbNameSuffix](#)  
*Widget name suffix for the thumb component.*

## Protected Member Functions

- virtual void [updateThumb](#) (void)  
*update the size and location of the thumb to properly represent the current state of the slider*
- virtual float [getValueFromThumb](#) (void) const  
*return value that best represents current slider value given the current location of the thumb.*
- virtual float [getAdjustDirectionFromPoint](#) (const [Point](#) &pt) const  
*Given window location pt, return a value indicating what change should be made to the slider.*
- bool [handleThumbMoved](#) (const [EventArgs](#) &e)  
*update the size and location of the thumb to properly represent the current state of the slider*

- bool [handleThumbTrackStarted](#) (const [EventArgs](#) &e)  
*handler function for when thumb tracking begins*
- bool [handleThumbTrackEnded](#) (const [EventArgs](#) &e)  
*handler function for when thumb tracking begins*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- virtual void [onValueChanged](#) ([WindowEventArgs](#) &e)  
*Handler triggered when the slider value changes.*
- virtual void [onThumbTrackStarted](#) ([WindowEventArgs](#) &e)  
*Handler triggered when the user begins to drag the slider thumb.*
- virtual void [onThumbTrackEnded](#) ([WindowEventArgs](#) &e)  
*Handler triggered when the slider thumb is released.*
- virtual void [onMouseButtonDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*

## Protected Attributes

- float [d\\_value](#)  
*current slider value*
- float [d\\_maxValue](#)  
*slider maximum value (minimum is fixed at 0)*
- float [d\\_step](#)  
*amount to adjust slider by when clicked (and not dragged).*

### 6.262.1 Detailed Description

Base class for [Slider](#) widgets.

The slider widget has a default range of 0.0f - 1.0f. This enables use of the slider value to scale any value needed by way of a simple multiplication.

## 6.262.2 Member Function Documentation

### 6.262.2.1 float CEGUI::Slider::getCurrentValue (void) const [inline]

return the current slider value.

#### Returns:

float value equal to the sliders current value.

### 6.262.2.2 float CEGUI::Slider::getMaxValue (void) const [inline]

return the maximum value set for this widget

#### Returns:

float value equal to the currently set maximum value for this slider.

### 6.262.2.3 float CEGUI::Slider::getClickStep (void) const [inline]

return the current click step setting for the slider.

The click step size is the amount the slider value will be adjusted when the widget is clicked wither side of the slider thumb.

#### Returns:

float value representing the current click step setting.

### 6.262.2.4 Thumb \* CEGUI::Slider::getThumb () const

Return a pointer to the [Thumb](#) component widget for this [Slider](#).

#### Returns:

Pointer to a [Thumb](#) object.

#### Exceptions:

[\*UnknownObjectException\*](#) Thrown if the [Thumb](#) component does not exist.

### 6.262.2.5 void CEGUI::Slider::initialiseComponents (void) [virtual]

Initialises the [Window](#) based object ready for use.

#### Note:

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

#### Returns:

Nothing

Reimplemented from [CEGUI::Window](#).

**6.262.2.6 void CEGUI::Slider::setMaxValue (float *maxVal*)**

set the maximum value for the slider. Note that the minimum value is fixed at 0.

**Parameters:**

*maxVal* float value specifying the maximum value for this slider widget.

**Returns:**

Nothing.

**6.262.2.7 void CEGUI::Slider::setCurrentValue (float *value*)**

set the current slider value.

**Parameters:**

*value* float value specifying the new value for this slider widget.

**Returns:**

Nothing.

**6.262.2.8 void CEGUI::Slider::setClickStep (float *step*) [inline]**

set the current click step setting for the slider.

The click step size is the amount the slider value will be adjusted when the widget is clicked wither side of the slider thumb.

**Parameters:**

*step* float value representing the click step setting to use.

**Returns:**

Nothing.

**6.262.2.9 float CEGUI::Slider::getValueFromThumb (void) const [protected, virtual]**

return value that best represents current slider value given the current location of the thumb.

**Returns:**

float value that, given the thumb widget position, best represents the current value for the slider.

**6.262.2.10 float CEGUI::Slider::getAdjustDirectionFromPoint (const Point & *pt*) const [protected, virtual]**

Given window location *pt*, return a value indicating what change should be made to the slider.

**Parameters:**

*pt* Point object describing a pixel position in window space.

**Returns:**

- -1 to indicate slider should be moved to a lower setting.
- 0 to indicate slider should not be moved.
- +1 to indicate slider should be moved to a higher setting.

**6.262.2.11 bool CEGUI::Slider::handleThumbMoved (const EventArgs & e) [protected]**

update the size and location of the thumb to properly represent the current state of the slider

return value that best represents current slider value given the current location of the thumb.

**Returns:**

float value that, given the thumb widget position, best represents the current value for the slider.

Given window location *pt*, return a value indicating what change should be made to the slider.

**Parameters:**

*pt* Point object describing a pixel position in window space.

**Returns:**

- -1 to indicate slider should be moved to a lower setting.
- 0 to indicate slider should not be moved.
- +1 to indicate slider should be moved to a higher setting.

handler function for when thumb moves.

**6.262.2.12 virtual bool CEGUI::Slider::testClassName\_impl (const String & class\_name) const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).



**6.262.2.13** `virtual bool CEGUI::Slider::validateWindowRenderer (const String & name) const`  
[inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.262.2.14** `void CEGUI::Slider::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.262.2.15** `void CEGUI::Slider::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

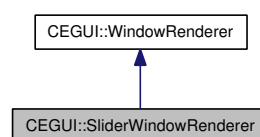
*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

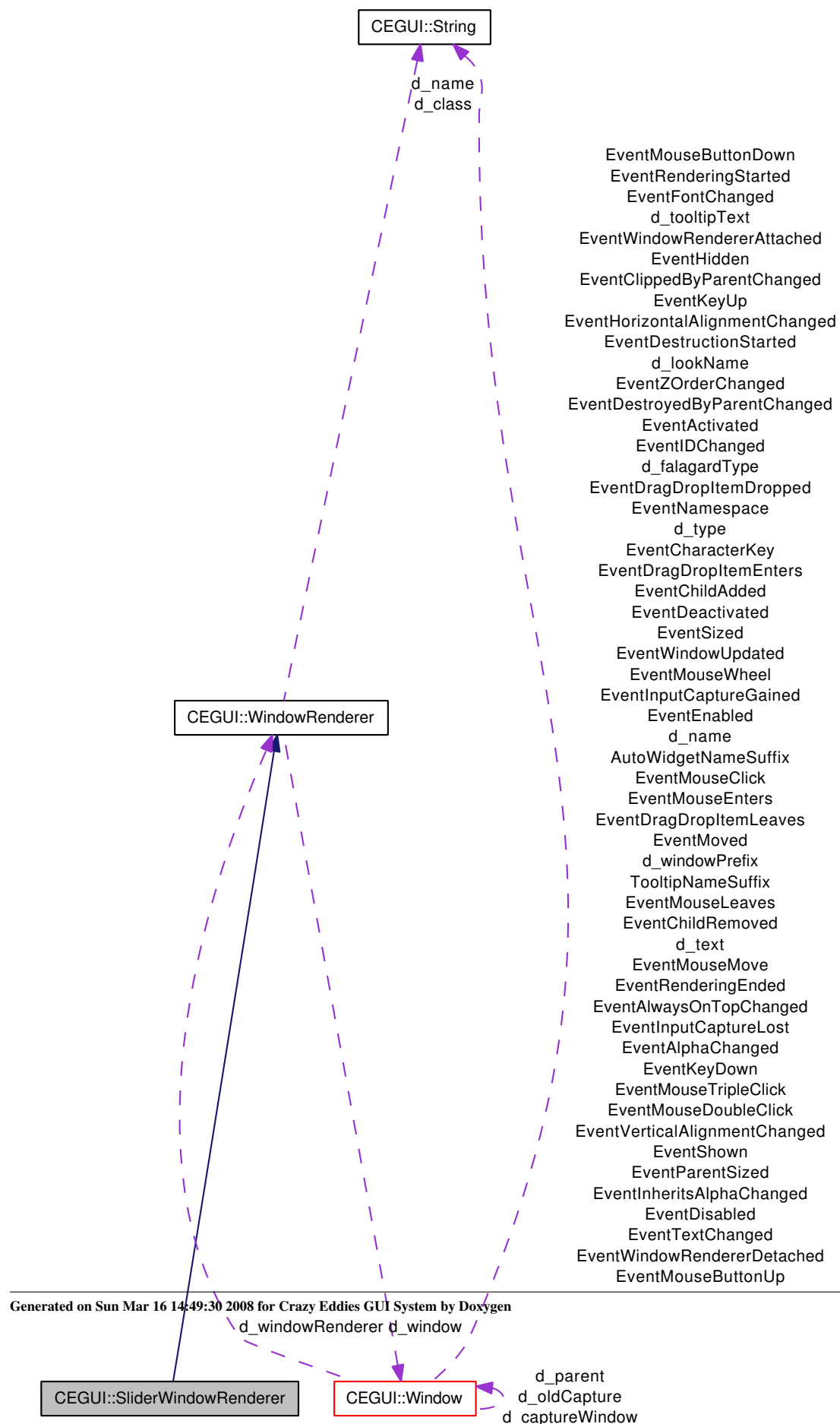
## 6.263 CEGUI::SliderWindowRenderer Class Reference

Base class for [ItemEntry](#) window renderer objects.

Inheritance diagram for CEGUI::SliderWindowRenderer:



Collaboration diagram for CEGUI::SliderWindowRenderer:



## Public Member Functions

- **SliderWindowRenderer** (const [String](#) &name)  
*Constructor.*
- virtual void **updateThumb** (void)=0  
*update the size and location of the thumb to properly represent the current state of the slider*
- virtual float **getValueFromThumb** (void) const =0  
*return value that best represents current slider value given the current location of the thumb.*
- virtual float **getAdjustDirectionFromPoint** (const [Point](#) &pt) const =0  
*Given window location pt, return a value indicating what change should be made to the slider.*

### 6.263.1 Detailed Description

Base class for [ItemEntry](#) window renderer objects.

### 6.263.2 Member Function Documentation

#### 6.263.2.1 virtual float CEGUI::SliderWindowRenderer::getValueFromThumb (void) const [pure virtual]

return value that best represents current slider value given the current location of the thumb.

##### Returns:

float value that, given the thumb widget position, best represents the current value for the slider.

#### 6.263.2.2 virtual float CEGUI::SliderWindowRenderer::getAdjustDirectionFromPoint (const [Point](#) & pt) const [pure virtual]

Given window location *pt*, return a value indicating what change should be made to the slider.

##### Parameters:

*pt* Point object describing a pixel position in window space.

##### Returns:

- -1 to indicate slider should be moved to a lower setting.
- 0 to indicate slider should not be moved.
- +1 to indicate slider should be moved to a higher setting.

## 6.264 CEGUI::SlotFunctorBase Class Reference

Defines abstract interface which will be used when constructing various functor objects that bind slots to signals (or in [CEGUI](#) terms, handlers to events).

Inherited by [CEGUI::FreeFunctionSlot](#), [CEGUI::FunctorCopySlot< T >](#), [CEGUI::FunctorPointerSlot< T >](#), [CEGUI::FunctorReferenceSlot< T >](#), and [CEGUI::MemberFunctionSlot< T >](#).

### Public Member Functions

- virtual bool **operator()** (const [EventArgs](#) &args)=0

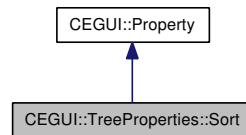
#### 6.264.1 Detailed Description

Defines abstract interface which will be used when constructing various functor objects that bind slots to signals (or in [CEGUI](#) terms, handlers to events).

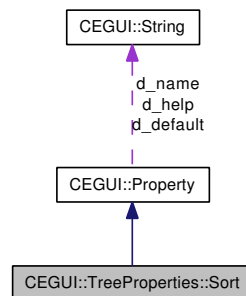
## 6.265 CEGUI::TreeProperties::Sort Class Reference

[Property](#) to access the sort setting of the list box.

Inheritance diagram for CEGUI::TreeProperties::Sort:



Collaboration diagram for CEGUI::TreeProperties::Sort:



### Public Member Functions

- `String get (const PropertyReceiver *receiver) const`  
*Return the current value of the [Property](#) as a [String](#).*
- `void set (PropertyReceiver *receiver, const String &value)`  
*Sets the value of the property.*

### 6.265.1 Detailed Description

[Property](#) to access the sort setting of the list box.

Usage:

- Name: [Sort](#)
- Format: "[text]"

Where [Text] is:

- "True" to indicate the list items should be sorted.
- "False" to indicate the list items should not be sorted.

## 6.265.2 Member Function Documentation

### 6.265.2.1 String CEGUI::TreeProperties::Sort::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.265.2.2 void CEGUI::TreeProperties::Sort::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

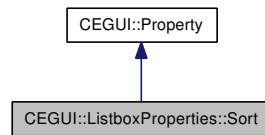
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

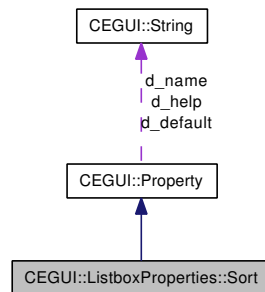
## 6.266 CEGUI::ListboxProperties::Sort Class Reference

[Property](#) to access the sort setting of the list box.

Inheritance diagram for CEGUI::ListboxProperties::Sort:



Collaboration diagram for CEGUI::ListboxProperties::Sort:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.266.1 Detailed Description

[Property](#) to access the sort setting of the list box.

**Usage:**

- Name: [Sort](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate the list items should be sorted.
- "False" to indicate the list items should not be sorted.



## 6.266.2 Member Function Documentation

### 6.266.2.1 String CEGUI::ListboxProperties::Sort::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.266.2.2 void CEGUI::ListboxProperties::Sort::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

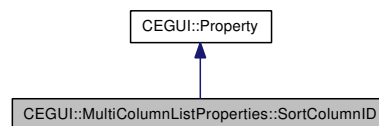
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

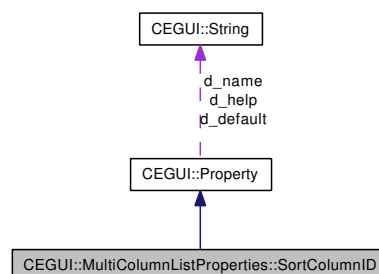
## 6.267 CEGUI::MultiColumnListProperties::SortColumnID Class Reference

[Property](#) to access the current sort column (via ID code).

Inheritance diagram for CEGUI::MultiColumnListProperties::SortColumnID:



Collaboration diagram for CEGUI::MultiColumnListProperties::SortColumnID:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.267.1 Detailed Description

[Property](#) to access the current sort column (via ID code).

#### Usage:

- Name: [SortColumnID](#)
- Format: "[uint]".

#### Where:

- [uint] is any unsigned integer value.

## 6.267.2 Member Function Documentation

### 6.267.2.1 String CEGUI::MultiColumnListProperties::SortColumnID::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.267.2.2 void CEGUI::MultiColumnListProperties::SortColumnID::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

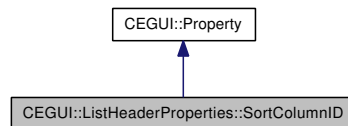
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

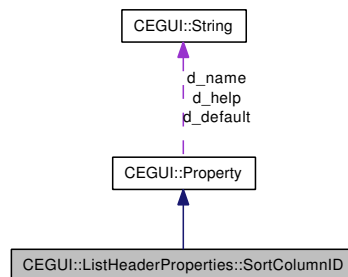
## 6.268 CEGUI::ListHeaderProperties::SortColumnID Class Reference

[Property](#) to access the current sort column (via ID code).

Inheritance diagram for CEGUI::ListHeaderProperties::SortColumnID:



Collaboration diagram for CEGUI::ListHeaderProperties::SortColumnID:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.268.1 Detailed Description

[Property](#) to access the current sort column (via ID code).

##### Usage:

- Name: [SortColumnID](#)
- Format: "[uint]".

##### Where:

- [uint] is any unsigned integer value.

## 6.268.2 Member Function Documentation

### 6.268.2.1 String CEGUI::ListHeaderProperties::SortColumnID::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.268.2.2 void CEGUI::ListHeaderProperties::SortColumnID::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

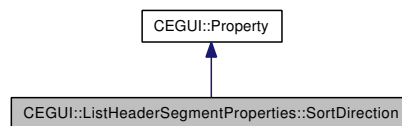
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

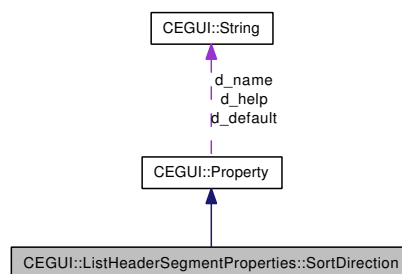
## 6.269 CEGUI::ListHeaderSegmentProperties::SortDirection Class Reference

[Property](#) to access the sort direction setting of the header segment.

Inheritance diagram for CEGUI::ListHeaderSegmentProperties::SortDirection:



Collaboration diagram for CEGUI::ListHeaderSegmentProperties::SortDirection:



### Public Member Functions

- `String` `get` (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

#### 6.269.1 Detailed Description

[Property](#) to access the sort direction setting of the header segment.

**Usage:**

- Name: [SortDirection](#)
- Format: "[text]"

Where [Text] is one of:

- "Ascending"
- "Descending"
- "None"

## 6.269.2 Member Function Documentation

### 6.269.2.1 String CEGUI::ListHeaderSegmentProperties::SortDirection::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.269.2.2 void CEGUI::ListHeaderSegmentProperties::SortDirection::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

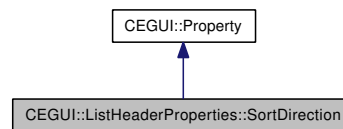
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

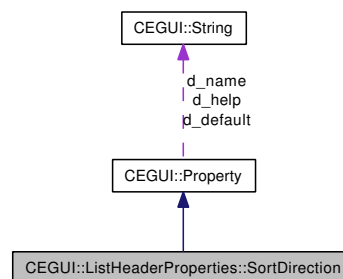
## 6.270 CEGUI::ListHeaderProperties::SortDirection Class Reference

[Property](#) to access the sort direction setting of the list header.

Inheritance diagram for CEGUI::ListHeaderProperties::SortDirection:



Collaboration diagram for CEGUI::ListHeaderProperties::SortDirection:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.270.1 Detailed Description

[Property](#) to access the sort direction setting of the list header.

Usage:

- Name: [SortDirection](#)
- Format: "[text]"

Where [Text] is one of:

- "Ascending"
- "Descending"
- "None"



## 6.270.2 Member Function Documentation

### 6.270.2.1 String CEGUI::ListHeaderProperties::SortDirection::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.270.2.2 void CEGUI::ListHeaderProperties::SortDirection::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

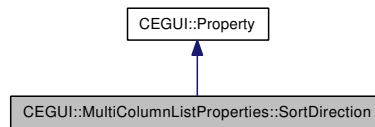
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

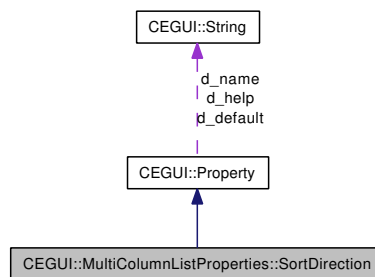
## 6.271 CEGUI::MultiColumnListProperties::SortDirection Class Reference

[Property](#) to access the sort direction setting of the list.

Inheritance diagram for CEGUI::MultiColumnListProperties::SortDirection:



Collaboration diagram for CEGUI::MultiColumnListProperties::SortDirection:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.271.1 Detailed Description

[Property](#) to access the sort direction setting of the list.

#### Usage:

- Name: [SortDirection](#)
- Format: "[text]"

Where [Text] is one of:

- "Ascending"
- "Descending"
- "None"

## 6.271.2 Member Function Documentation

### 6.271.2.1 String CEGUI::MultiColumnListProperties::SortDirection::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.271.2.2 void CEGUI::MultiColumnListProperties::SortDirection::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

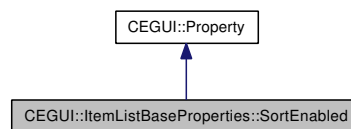
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

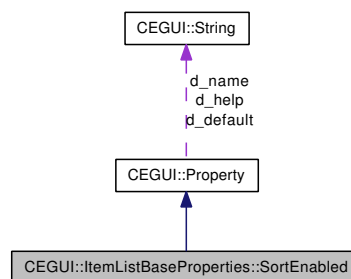
## 6.272 CEGUI::ItemListBaseProperties::SortEnabled Class Reference

[Property](#) to access the state of the sorting enabled setting.

Inheritance diagram for CEGUI::ItemListBaseProperties::SortEnabled:



Collaboration diagram for CEGUI::ItemListBaseProperties::SortEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.272.1 Detailed Description

[Property](#) to access the state of the sorting enabled setting.

Usage:

- Name: [SortEnabled](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate that sorting is enabled.
- "False" to indicate that sorting is disabled.

## 6.272.2 Member Function Documentation

### 6.272.2.1 String CEGUI::ItemListBaseProperties::SortEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.272.2.2 void CEGUI::ItemListBaseProperties::SortEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

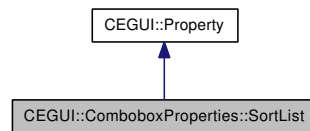
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

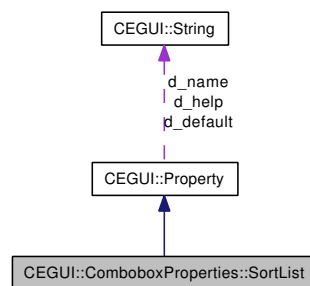
## 6.273 CEGUI::ComboboxProperties::SortList Class Reference

[Property](#) to access the sort setting of the list box.

Inheritance diagram for CEGUI::ComboboxProperties::SortList:



Collaboration diagram for CEGUI::ComboboxProperties::SortList:



### Public Member Functions

- **String** **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const **String** &value)  
*Sets the value of the property.*

### 6.273.1 Detailed Description

[Property](#) to access the sort setting of the list box.

**Usage:**

- Name: [SortList](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate the list items should be sorted.
- "False" to indicate the list items should not be sorted.

## 6.273.2 Member Function Documentation

### 6.273.2.1 String CEGUI::ComboboxProperties::SortList::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.273.2.2 void CEGUI::ComboboxProperties::SortList::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

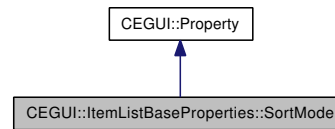
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

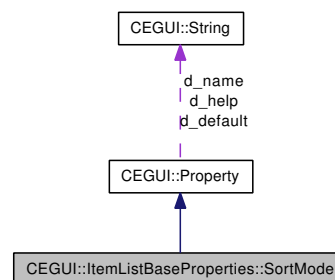
## 6.274 CEGUI::ItemListBaseProperties::SortMode Class Reference

[Property](#) to access the sorting mode.

Inheritance diagram for CEGUI::ItemListBaseProperties::SortMode:



Collaboration diagram for CEGUI::ItemListBaseProperties::SortMode:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.274.1 Detailed Description

[Property](#) to access the sorting mode.

**Usage:**

- Name: [SortMode](#)
- Format: "[text]".

**Where [Text] is:**

- "Ascending" to use ascending sorting.
- "Descending" to use descending sorting.
- "UserSort" to use a user specified callback as sorting function, defaults to ascending sorting if no user callback has been specified by the application.



## 6.274.2 Member Function Documentation

### 6.274.2.1 String CEGUI::ItemListBaseProperties::SortMode::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.274.2.2 void CEGUI::ItemListBaseProperties::SortMode::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

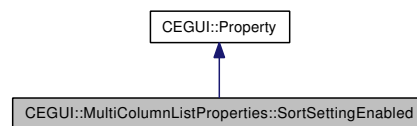
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

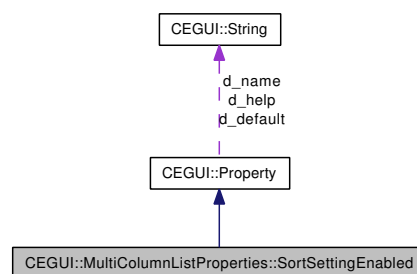
## 6.275 CEGUI::MultiColumnListProperties::SortSettingEnabled Class Reference

[Property](#) to access the setting for user modification of the sort column & direction.

Inheritance diagram for CEGUI::MultiColumnListProperties::SortSettingEnabled:



Collaboration diagram for CEGUI::MultiColumnListProperties::SortSettingEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.275.1 Detailed Description

[Property](#) to access the setting for user modification of the sort column & direction.

**Usage:**

- Name: [SortSettingEnabled](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate the user may modify the sort column and direction by clicking the header segments.
- "False" to indicate the user may not modify the sort column or direction.

## 6.275.2 Member Function Documentation

### 6.275.2.1 String CEGUI::MultiColumnListProperties::SortSettingEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.275.2.2 void CEGUI::MultiColumnListProperties::SortSettingEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

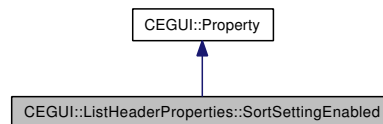
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

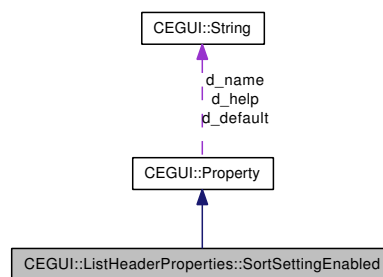
## 6.276 CEGUI::ListHeaderProperties::SortSettingEnabled Class Reference

[Property](#) to access the setting for user modification of the sort column & direction.

Inheritance diagram for CEGUI::ListHeaderProperties::SortSettingEnabled:



Collaboration diagram for CEGUI::ListHeaderProperties::SortSettingEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.276.1 Detailed Description

[Property](#) to access the setting for user modification of the sort column & direction.

#### Usage:

- Name: [SortSettingEnabled](#)
- Format: "[text]"

#### Where [Text] is:

- "True" to indicate the user may modify the sort column and direction by clicking the header segments.
- "False" to indicate the user may not modify the sort column or direction.

## 6.276.2 Member Function Documentation

### 6.276.2.1 String CEGUI::ListHeaderProperties::SortSettingEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.276.2.2 void CEGUI::ListHeaderProperties::SortSettingEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

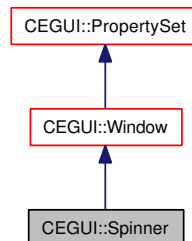
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

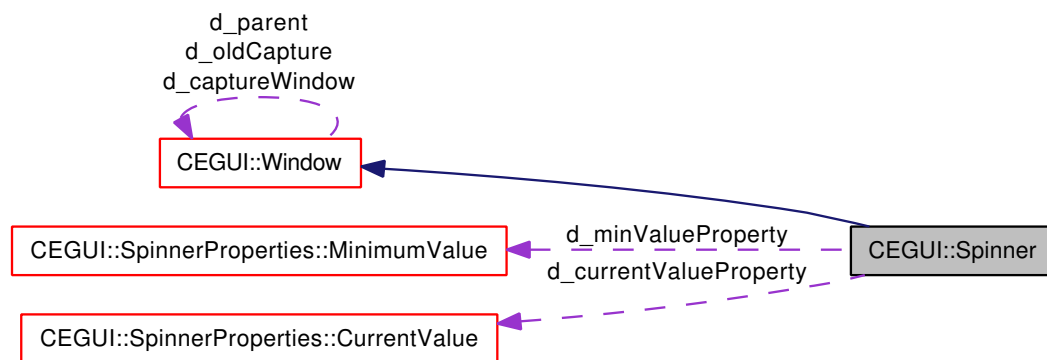
## 6.277 CEGUI::Spinner Class Reference

Base class for the [Spinner](#) widget.

Inheritance diagram for CEGUI::Spinner:



Collaboration diagram for CEGUI::Spinner:



### Public Types

- enum [TextInputMode](#) { [FloatingPoint](#), [Integer](#), [Hexadecimal](#), [Octal](#) }  
*Enumerated type specifying possible input and/or display modes for the spinner.*

### Public Member Functions

- [Spinner](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Spinner](#) objects.*
- virtual [~Spinner](#) (void)  
*Destructor for [Spinner](#) objects.*
- void [initialiseComponents](#) (void)  
*Initialises the [Window](#) based object ready for use.*
- float [getCurrentValue](#) (void) const  
*Return the current spinner value.*

- float [getStepSize](#) (void) const  
*Return the current step value.*
- float [getMaximumValue](#) (void) const  
*Return the current maximum limit value for the [Spinner](#).*
- float [getMinimumValue](#) (void) const  
*Return the current minimum limit value for the [Spinner](#).*
- [TextInputMode](#) [getTextInputMode](#) (void) const  
*Return the current text input / display mode setting.*
- void [setCurrentValue](#) (float value)  
*Set the current spinner value.*
- void [setStepSize](#) (float step)  
*Set the current step value.*
- void [setMaximumValue](#) (float maxVaue)  
*Set the spinner maximum value.*
- void [setMinimumValue](#) (float minVaue)  
*Set the spinner minimum value.*
- void [setTextInputMode](#) ([TextInputMode](#) mode)  
*Set the spinner input / display mode.*

## Static Public Attributes

- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [EventValueChanged](#)  
*[Event](#) fired when the spinner value changes.*
- static const [String](#) [EventStepChanged](#)  
*[Event](#) fired when the step value changes.*
- static const [String](#) [EventMaximumValueChanged](#)  
*[Event](#) fired when the maximum spinner value changes.*
- static const [String](#) [EventMinimumValueChanged](#)  
*[Event](#) fired when the minimum spinner value changes.*

- static const [String EventTextInputModeChanged](#)  
*Event fired when the input/display mode is changed.*
- static const [String EditboxNameSuffix](#)  
*Widget name suffix for the editbox thumb component.*
- static const [String IncreaseButtonNameSuffix](#)  
*Widget name suffix for the increase button component.*
- static const [String DecreaseButtonNameSuffix](#)  
*Widget name suffix for the decrease button component.*

## Protected Member Functions

- virtual float [getValueFromText](#) (void) const  
*Returns the numerical representation of the current editbox text.*
- virtual [String getTextFromValue](#) (void) const  
*Returns the textual representation of the current spinner value.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- [PushButton](#) \* [getIncreaseButton](#) () const  
*Return a pointer to the 'increase' PushButtoncomponent widget for this [Spinner](#).*
- [PushButton](#) \* [getDecreaseButton](#) () const  
*Return a pointer to the 'decrease' [PushButton](#) component widget for this [Spinner](#).*
- [Editbox](#) \* [getEditbox](#) () const  
*Return a pointer to the [Editbox](#) component widget for this [Spinner](#).*
- virtual void [onFontChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's font is changed.*
- virtual void [onTextChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's text is changed.*
- virtual void [onActivated](#) ([ActivationEventArgs](#) &e)  
*Handler called when this window has become the active window.*
- virtual void [onValueChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the spinner value changes.*
- virtual void [onStepChanged](#) ([WindowEventArgs](#) &e)  
*Method called when the step value changes.*
- virtual void [onMaximumValueChanged](#) ([WindowEventArgs](#) &e)



*Method called when the maximum value setting changes.*

- virtual void [onMinimumValueChanged](#) (WindowEventArgs &e)

*Method called when the minimum value setting changes.*

- virtual void [onTextInputModeChanged](#) (WindowEventArgs &e)

*Method called when the text input/display mode is changed.*

- bool **handleIncreaseButton** (const EventArgs &e)
- bool **handleDecreaseButton** (const EventArgs &e)
- bool **handleEditTextChange** (const EventArgs &e)

## Protected Attributes

- float [d\\_stepSize](#)

*Step size value used y the increase & decrease buttons.*

- float [d\\_currentValue](#)

*Numerical copy of the text in d\_editbox.*

- float [d\\_maxValue](#)

*Maximum value for spinner.*

- float [d\\_minValue](#)

*Minimum value for spinner.*

- [TextInputMode](#) [d\\_inputMode](#)

*Current text display/input mode.*

## Static Protected Attributes

- static const [String](#) [FloatValidator](#)

*Validator regex used for floating point mode.*

- static const [String](#) [IntegerValidator](#)

*Validator regex used for decimal integer mode.*

- static const [String](#) [HexValidator](#)

*Validator regex used for hexadecimal mode.*

- static const [String](#) [OctalValidator](#)

*Validator regex used for octal mode.*

### 6.277.1 Detailed Description

Base class for the [Spinner](#) widget.

The spinner widget has a text area where numbers may be entered and two buttons which may be used to increase or decrease the value in the text area by a user specified amount.

**Note:**

While the [Spinner](#) widget has support for floating point values, the results of using this support in its current state may not be satisfactory. The general advice, for the moment, is to avoid very large or very small values in floating point mode, and to perform as little manipulation of the values as possible. The various issues you may see range from scientific notation appearing in the box, to complete breakdown of 'expected' values upon manipulation. This is something that we intend to address for a future release.

### 6.277.2 Member Enumeration Documentation

#### 6.277.2.1 enum CEGUI::Spinner::TextInputMode

Enumerated type specifying possible input and/or display modes for the spinner.

**Enumerator:**

***FloatingPoint*** Floating point decimal.

***Integer*** Integer decimal.

***Hexadecimal*** Hexadecimal.

***Octal*** Octal.

### 6.277.3 Member Function Documentation

#### 6.277.3.1 void CEGUI::Spinner::initialiseComponents (void) [virtual]

Initialises the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

#### 6.277.3.2 float CEGUI::Spinner::getCurrentValue (void) const

Return the current spinner value.

**Returns:**

current float value of the [Spinner](#).

**6.277.3.3 float CEGUI::Spinner::getStepSize (void) const**

Return the current step value.

**Returns:**

float step value. This is the value added to the spinner vaue when the up / down buttons are clicked.

**6.277.3.4 float CEGUI::Spinner::getMaximumValue (void) const**

Return the current maximum limit value for the [Spinner](#).

**Returns:**

Maximum value that is allowed for the spinner.

**6.277.3.5 float CEGUI::Spinner::getMinimumValue (void) const**

Return the current minimum limit value for the [Spinner](#).

**Returns:**

Minimum value that is allowed for the spinner.

**6.277.3.6 Spinner::TextInputMode CEGUI::Spinner::getTextInputMode (void) const**

Return the current text input / display mode setting.

**Returns:**

One of the TextInputMode enumerated values indicating the current text input and display mode.

**6.277.3.7 void CEGUI::Spinner::setCurrentValue (float *value*)**

Set the current spinner value.

**Parameters:**

*value* value to be assigned to the [Spinner](#).

**Returns:**

Nothing.

**6.277.3.8 void CEGUI::Spinner::setStepSize (float *step*)**

Set the current step value.

**Parameters:**

*step* The value added to be the spinner value when the up / down buttons are clicked.

**Returns:**

Nothing.

**6.277.3.9 void CEGUI::Spinner::setMaximumValue (float *maxValue*)**

Set the spinner maximum value.

**Parameters:**

*maxValue* The maximum value to be allowed by the spinner.

**Returns:**

Nothing.

**6.277.3.10 void CEGUI::Spinner::setMinimumValue (float *minVaue*)**

Set the spinner minimum value.

**Parameters:**

*minVaue* The minimum value to be allowed by the spinner.

**Returns:**

Nothing.

**6.277.3.11 void CEGUI::Spinner::setTextInputMode (TextInputMode *mode*)**

Set the spinner input / display mode.

**Parameters:**

*mode* One of the TextInputMode enumerated values indicating the text input / display mode to be used by the spinner.

**Returns:**

Nothing.

**6.277.3.12 float CEGUI::Spinner::getValueFromText (void) const** [protected, virtual]

Returns the numerical representation of the current editbox text.

**Returns:**

float value that is the numerical equivalent of the editbox text.

**Exceptions:**

[\*InvalidRequestException\*](#) thrown if the text can not be converted.

**6.277.3.13 String CEGUI::Spinner::getTextFromValue (void) const** [protected, virtual]

Returns the textual representation of the current spinner value.

**Returns:**

[String](#) object that is equivalent to the the numerical value of the spinner.

**6.277.3.14 virtual bool CEGUI::Spinner::testClassName\_impl (const String & class\_name) const**  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.277.3.15 PushButton \* CEGUI::Spinner::getIncreaseButton () const** [protected]

Return a pointer to the 'increase' PushButtoncomponent widget for this [Spinner](#).

**Returns:**

Pointer to a [PushButton](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the increase [PushButton](#) component does not exist.

**6.277.3.16** `PushButton * CEGUI::Spinner::getDecreaseButton () const` [protected]

Return a pointer to the 'decrease' [PushButton](#) component widget for this [Spinner](#).

**Returns:**

Pointer to a [PushButton](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the 'decrease' [PushButton](#) component does not exist.

**6.277.3.17** `Editbox * CEGUI::Spinner::getEditbox () const` [protected]

Return a pointer to the [Editbox](#) component widget for this [Spinner](#).

**Returns:**

Pointer to a [Editbox](#) object.

**Exceptions:**

[\*UnknownObjectException\*](#) Thrown if the [Editbox](#) component does not exist.

**6.277.3.18** `void CEGUI::Spinner::onFontChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's font is changed.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.277.3.19** `void CEGUI::Spinner::onTextChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.277.3.20** `void CEGUI::Spinner::onActivated (ActivationEventArgs & e)` [protected, virtual]

Handler called when this window has become the active window.

**Parameters:**

*e* [ActivationEventArgs](#) class whose 'otherWindow' field is set to the window that previously was active, or NULL for none.

Reimplemented from [CEGUI::Window](#).

**6.277.3.21** `void CEGUI::Spinner::onValueChanged (WindowEventArgs & e)` [protected, virtual]

Method called when the spinner value changes.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.277.3.22** `void CEGUI::Spinner::onStepChanged (WindowEventArgs & e)` [protected, virtual]

Method called when the step value changes.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.277.3.23** `void CEGUI::Spinner::onMaximumValueChanged (WindowEventArgs & e)`  
[protected, virtual]

Method called when the maximum value setting changes.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.277.3.24 void CEGUI::Spinner::onMinimumValueChanged (WindowEventArgs & *e*)**  
[protected, virtual]

Method called when the minimum value setting changes.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

Nothing.

**6.277.3.25 void CEGUI::Spinner::onTextInputModeChanged (WindowEventArgs & *e*)**  
[protected, virtual]

Method called when the text input/display mode is changed.

**Parameters:**

*e* [WindowEventArgs](#) object containing any relevant data.

**Returns:**

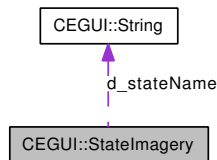
Nothing.



## 6.278 CEGUI::StateImagery Class Reference

Class the encapsulates imagery for a given widget state.

Collaboration diagram for CEGUI::StateImagery:



### Public Member Functions

- [StateImagery](#) ()  
*Constructor.*
- [StateImagery](#) (const [String](#) &name)  
*Constructor.*
- void [render](#) ([Window](#) &srcWindow, const [ColourRect](#) \*modcols=0, const [Rect](#) \*clipper=0) const  
*Render imagery for this state.*
- void [render](#) ([Window](#) &srcWindow, const [Rect](#) &baseRect, const [ColourRect](#) \*modcols=0, const [Rect](#) \*clipper=0) const  
*Render imagery for this state.*
- void [addLayer](#) (const [LayerSpecification](#) &layer)  
*Add an imagery LayerSpecification to this state.*
- void [clearLayers](#) ()  
*Removed all LayerSpecifications from this state.*
- const [String](#) & [getName](#) () const  
*Return the name of this state.*
- bool [isClippedToDisplay](#) () const  
*Return whether this state imagery should be clipped to the display rather than the target window.*
- void [setClippedToDisplay](#) (bool setting)  
*Set whether this state imagery should be clipped to the display rather than the target window.*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this StateImagery to out\_stream.*

### 6.278.1 Detailed Description

Class the encapsulates imagery for a given widget state.

## 6.278.2 Constructor & Destructor Documentation

### 6.278.2.1 CEGUI::StateImagery::StateImagery (const String & *name*)

Constructor.

**Parameters:**

*name* Name of the state

## 6.278.3 Member Function Documentation

### 6.278.3.1 void CEGUI::StateImagery::render (Window & *srcWindow*, const ColourRect \* *modcols* = 0, const Rect \* *clipper* = 0) const

Render imagery for this state.

**Parameters:**

*srcWindow* [Window](#) to use when convering [BaseDim](#) values to pixels.

**Returns:**

Nothing.

### 6.278.3.2 void CEGUI::StateImagery::render (Window & *srcWindow*, const Rect & *baseRect*, const ColourRect \* *modcols* = 0, const Rect \* *clipper* = 0) const

Render imagery for this state.

**Parameters:**

*srcWindow* [Window](#) to use when convering [BaseDim](#) values to pixels.

*baseRect* [Rect](#) to use when convering [BaseDim](#) values to pixels.

**Returns:**

Nothing.

### 6.278.3.3 void CEGUI::StateImagery::addLayer (const LayerSpecification & *layer*)

Add an imagery [LayerSpecification](#) to this state.

**Parameters:**

*layer* [LayerSpecification](#) to be added to this state (will be copied)

**Returns:**

Nothing.

**6.278.3.4 void CEGUI::StateImagery::clearLayers ()**

Removed all LayerSpecifications from this state.

**Returns:**

Nothing.

**6.278.3.5 const String & CEGUI::StateImagery::getName () const**

Return the name of this state.

**Returns:**

[String](#) object holding the name of the [StateImagery](#) object.

**6.278.3.6 bool CEGUI::StateImagery::isClippedToDisplay () const**

Return whether this state imagery should be clipped to the display rather than the target window.

Clipping to the display effectively implies that the imagery should be rendered unclipped.

/return

- true if the imagery will be clipped to the display area.
- false if the imagery will be clipped to the target window area.

**6.278.3.7 void CEGUI::StateImagery::setClippedToDisplay (bool *setting*)**

Set whether this state imagery should be clipped to the display rather than the target window.

Clipping to the display effectively implies that the imagery should be rendered unclipped.

**Parameters:**

- setting*
- true if the imagery should be clipped to the display area.
  - false if the imagery should be clipped to the target window area.

**Returns:**

Nothing.

**6.278.3.8 void CEGUI::StateImagery::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [StateImagery](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

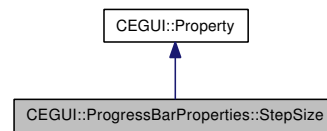
**Returns:**

Nothing.

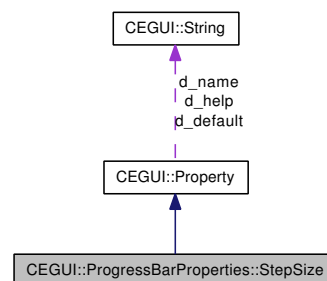
## 6.279 CEGUI::ProgressBarProperties::StepSize Class Reference

[Property](#) to access the step size setting for the progress bar.

Inheritance diagram for CEGUI::ProgressBarProperties::StepSize:



Collaboration diagram for CEGUI::ProgressBarProperties::StepSize:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.279.1 Detailed Description

[Property](#) to access the step size setting for the progress bar.

Usage:

- Name: [StepSize](#)
- Format: "[float]".

Where:

- [float] is the size of the invisible sizing border in screen pixels.

## 6.279.2 Member Function Documentation

### 6.279.2.1 String CEGUI::ProgressBarProperties::StepSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.279.2.2 void CEGUI::ProgressBarProperties::StepSize::set (PropertyReceiver \* *receiver*, const [String](#) & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

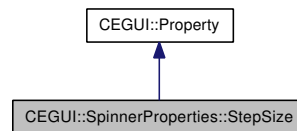
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

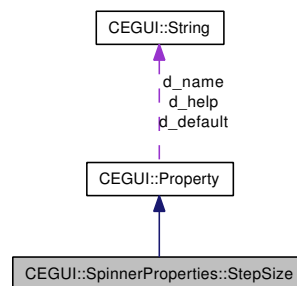
## 6.280 CEGUI::SpinnerProperties::StepSize Class Reference

[Property](#) to access the step size of the spinner.

Inheritance diagram for CEGUI::SpinnerProperties::StepSize:



Collaboration diagram for CEGUI::SpinnerProperties::StepSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.280.1 Detailed Description

[Property](#) to access the step size of the spinner.

#### Usage:

- Name: [StepSize](#)
- Format: "[float]".

#### Where:

- [float] represents the current value of the [Spinner](#) widget.

## 6.280.2 Member Function Documentation

### 6.280.2.1 String CEGUI::SpinnerProperties::StepSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.280.2.2 void CEGUI::SpinnerProperties::StepSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

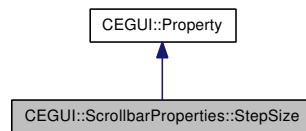
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

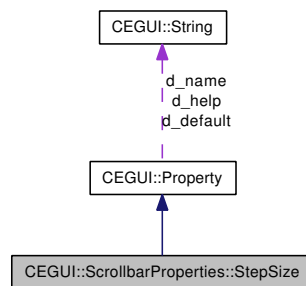
## 6.281 CEGUI::ScrollbarProperties::StepSize Class Reference

[Property](#) to access the step size for the [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollbarProperties::StepSize:



Collaboration diagram for CEGUI::ScrollbarProperties::StepSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.281.1 Detailed Description

[Property](#) to access the step size for the [Scrollbar](#).

#### Usage:

- Name: [StepSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the increase/decrease button step (as defined by the client code).



## 6.281.2 Member Function Documentation

### 6.281.2.1 String CEGUI::ScrollbarProperties::StepSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.281.2.2 void CEGUI::ScrollbarProperties::StepSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.282 CEGUI::String Class Reference

[String](#) class used within the GUI system.

### Public Types

- typedef utf32 [value\\_type](#)  
*Basic 'code point' type used for [String](#) (utf32).*
- typedef size\_t [size\\_type](#)  
*Unsigned type used for size values and indices.*
- typedef ptrdiff\_t [difference\\_type](#)  
*Signed type used for differences.*
- typedef utf32 & [reference](#)  
*Type used for utf32 code point references.*
- typedef const utf32 & [const\\_reference](#)  
*Type used for constant utf32 code point references.*
- typedef utf32 \* [pointer](#)  
*Type used for utf32 code point pointers.*
- typedef const utf32 \* [const\\_pointer](#)  
*Type used for constant utf32 code point pointers.*
- typedef std::reverse\_iterator< [const\\_iterator](#) > [const\\_reverse\\_iterator](#)  
*Constant reverse [iterator](#) class for [String](#) objects.*
- typedef std::reverse\_iterator< [iterator](#) > [reverse\\_iterator](#)  
*Reverse [iterator](#) class for [String](#) objects.*

### Public Member Functions

- [String](#) (void)  
*Constructs an empty string.*
- [~String](#) (void)  
*Destructor for [String](#) objects.*
- [String](#) (const [String](#) &str)  
*Copy constructor - Creates a new string with the same value as str.*
- [String](#) (const [String](#) &str, [size\\_type](#) str\_idx, [size\\_type](#) str\_num=[npos](#))  
*Constructs a new string initialised with code points from another [String](#) object.*
- [String](#) (const std::string &std\_str)

*Constructs a new string and initialises it using the std::string std\_str.*

- **String** (const std::string &std\_str, **size\_type** str\_idx, **size\_type** str\_num=npos)  
*Constructs a new string initialised with characters from the given std::string object.*
- **String** (const utf8 \*utf8\_str)  
*Constructs a new **String** object and initialise it using the provided utf8 encoded string buffer.*
- **String** (const utf8 \*utf8\_str, **size\_type** chars\_len)  
*Constructs a new **String** object and initialise it using the provided utf8 encoded string buffer.*
- **String** (**size\_type** num, utf32 code\_point)  
*Constructs a new **String** that is initialised with the specified code point.*
- **String** (const\_iterator iter\_beg, const\_iterator iter\_end)  
*Construct a new string object and initialise it with code-points from the range [beg, end).*
- **String** (const char \*cstr)  
*Constructs a new **String** object and initialise it using the provided c-string.*
- **String** (const char \*chars, **size\_type** chars\_len)  
*Constructs a new **String** object and initialise it using characters from the provided char array.*
- **size\_type** size (void) const  
*Returns the size of the **String** in code points.*
- **size\_type** length (void) const  
*Returns the size of the **String** in code points.*
- bool empty (void) const  
*Returns true if the **String** is empty.*
- **size\_type** capacity (void) const  
*Return the number of code points that the **String** could hold before a re-allocation would be required.*
- void reserve (**size\_type** num=0)  
*Specifies the amount of reserve capacity to allocate.*
- int compare (const **String** &str) const  
*Compares this **String** with the **String** 'str'.*
- int compare (**size\_type** idx, **size\_type** len, const **String** &str, **size\_type** str\_idx=0, **size\_type** str\_len=npos) const  
*Compares code points from this **String** with code points from the **String** 'str'.*
- int compare (const std::string &std\_str) const  
*Compares this **String** with the std::string 'std\_str'.*
- int compare (**size\_type** idx, **size\_type** len, const std::string &std\_str, **size\_type** str\_idx=0, **size\_type** str\_len=npos) const

Compares code points from this *String* with code points from the `std::string 'std_str'`.

- `int compare (const utf8 *utf8_str) const`  
Compares this *String* with the null-terminated utf8 encoded '`utf8_str`'.
- `int compare (size_type idx, size_type len, const utf8 *utf8_str) const`  
Compares code points from this *String* with the null-terminated utf8 encoded '`utf8_str`'.
- `int compare (size_type idx, size_type len, const utf8 *utf8_str, size_type str_cplen) const`  
Compares code points from this *String* with the utf8 encoded data in buffer '`utf8_str`'.
- `int compare (const char *cstr) const`  
Compares this *String* with the given c-string.
- `int compare (size_type idx, size_type len, const char *cstr) const`  
Compares code points from this *String* with the given c-string.
- `int compare (size_type idx, size_type len, const char *chars, size_type chars_len) const`  
Compares code points from this *String* with chars in the given char array.
- `reference operator[] (size_type idx)`  
Returns the code point at the given index.
- `value_type operator[] (size_type idx) const`  
Returns the code point at the given index.
- `reference at (size_type idx)`  
Returns the code point at the given index.
- `const_reference at (size_type idx) const`  
Returns the code point at the given index.
- `const char * c_str (void) const`  
Returns contents of the *String* as a null terminated string of utf8 encoded data.
- `const utf8 * data (void) const`  
Returns contents of the *String* as utf8 encoded data.
- `utf32 * ptr (void)`  
Returns a pointer to the buffer in use.
- `const utf32 * ptr (void) const`  
Returns a pointer to the buffer in use. (const version).
- `size_type copy (utf8 *buf, size_type len=npos, size_type idx=0) const`  
Copies an area of the *String* into the provided buffer as encoded utf8 data.
- `size_type utf8_stream_len (size_type num=npos, size_type idx=0) const`  
Return the number of utf8 code units required to hold an area of the *String* when encoded as utf8 data.

- **String & operator=** (const **String** &str)  
*Assign the value of **String** str to this **String**.*
- **String & assign** (const **String** &str, **size\_type** str\_idx=0, **size\_type** str\_num=npos)  
*Assign a sub-string of **String** str to this **String**.*
- **String & operator=** (const std::string &std\_str)  
*Assign the value of std::string std\_str to this **String**.*
- **String & assign** (const std::string &std\_str, **size\_type** str\_idx=0, **size\_type** str\_num=npos)  
*Assign a sub-string of std::string std\_str to this **String**.*
- **String & operator=** (const utf8 \*utf8\_str)  
*Assign to this **String** the string value represented by the given null-terminated utf8 encoded data.*
- **String & assign** (const utf8 \*utf8\_str)  
*Assign to this **String** the string value represented by the given null-terminated utf8 encoded data.*
- **String & assign** (const utf8 \*utf8\_str, **size\_type** str\_num)  
*Assign to this **String** the string value represented by the given utf8 encoded data.*
- **String & operator=** (utf32 code\_point)  
*Assigns the specified utf32 code point to this **String**. Result is always a **String** 1 code point in length.*
- **String & assign** (**size\_type** num, utf32 code\_point)  
*Assigns the specified code point repeatedly to the **String**.*
- **String & operator=** (const char \*cstr)  
*Assign to this **String** the given C-string.*
- **String & assign** (const char \*cstr)  
*Assign to this **String** the given C-string.*
- **String & assign** (const char \*chars, **size\_type** chars\_len)  
*Assign to this **String** a number of chars from a char array.*
- void **swap** (**String** &str)  
*Swaps the value of this **String** with the given **String** str.*
- **String & operator+=** (const **String** &str)  
*Appends the **String** str.*
- **String & append** (const **String** &str, **size\_type** str\_idx=0, **size\_type** str\_num=npos)  
*Appends a sub-string of the **String** str.*
- **String & operator+=** (const std::string &std\_str)  
*Appends the std::string std\_str.*
- **String & append** (const std::string &std\_str, **size\_type** str\_idx=0, **size\_type** str\_num=npos)  
*Appends a sub-string of the std::string std\_str.*

- [String](#) & [operator+=](#) (const utf8 \*utf8\_str)  
*Appends to the [String](#) the null-terminated utf8 encoded data in the buffer utf8\_str.*
- [String](#) & [append](#) (const utf8 \*utf8\_str)  
*Appends to the [String](#) the null-terminated utf8 encoded data in the buffer utf8\_str.*
- [String](#) & [append](#) (const utf8 \*utf8\_str, [size\\_type](#) len)  
*Appends to the [String](#) the utf8 encoded data in the buffer utf8\_str.*
- [String](#) & [operator+=](#) (utf32 code\_point)  
*Appends a single code point to the string.*
- [String](#) & [append](#) ([size\\_type](#) num, utf32 code\_point)  
*Appends a single code point multiple times to the string.*
- void [push\\_back](#) (utf32 code\_point)  
*Appends a single code point to the string.*
- [String](#) & [append](#) (const\_iterator iter\_beg, const\_iterator iter\_end)  
*Appends the code points in the reange [beg, end).*
- [String](#) & [operator+=](#) (const char \*cstr)  
*Appends to the [String](#) the given c-string.*
- [String](#) & [append](#) (const char \*cstr)  
*Appends to the [String](#) the given c-string.*
- [String](#) & [append](#) (const char \*chars, [size\\_type](#) chars\_len)  
*Appends to the [String](#) chars from the given char array.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const [String](#) &str)  
*Inserts the given [String](#) object at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const [String](#) &str, [size\\_type](#) str\_idx, [size\\_type](#) str\_num)  
*Inserts a sub-string of the given [String](#) object at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const std::string &std\_str)  
*Inserts the given std::string object at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const std::string &std\_str, [size\\_type](#) str\_idx, [size\\_type](#) str\_num)  
*Inserts a sub-string of the given std::string object at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const utf8 \*utf8\_str)  
*Inserts the given null-terminated utf8 encoded data at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const utf8 \*utf8\_str, [size\\_type](#) len)  
*Inserts the given utf8 encoded data at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, [size\\_type](#) num, utf32 code\_point)

*Inserts a code point multiple times into the [String](#).*

- void [insert](#) ([iterator](#) pos, [size\\_type](#) num, utf32 code\_point)  
*Inserts a code point multiple times into the [String](#).*
- [iterator](#) [insert](#) ([iterator](#) pos, utf32 code\_point)  
*Inserts a single code point into the [String](#).*
- void [insert](#) ([iterator](#) iter\_pos, [const\\_iterator](#) iter\_beg, [const\\_iterator](#) iter\_end)  
*Inserts code points specified by the range [beg, end).*
- [String](#) & [insert](#) ([size\\_type](#) idx, const char \*cstr)  
*Inserts the given c-string at the specified position.*
- [String](#) & [insert](#) ([size\\_type](#) idx, const char \*chars, [size\\_type](#) chars\_len)  
*Inserts chars from the given char array at the specified position.*
- void [clear](#) (void)  
*Removes all data from the [String](#).*
- [String](#) & [erase](#) (void)  
*Removes all data from the [String](#).*
- [String](#) & [erase](#) ([size\\_type](#) idx)  
*Erase a single code point from the string.*
- [String](#) & [erase](#) ([size\\_type](#) idx, [size\\_type](#) len=npos)  
*Erase a range of code points.*
- [String](#) & [erase](#) ([iterator](#) pos)  
*Erase the code point described by the given [iterator](#).*
- [String](#) & [erase](#) ([iterator](#) iter\_beg, [iterator](#) iter\_end)  
*Erase a range of code points described by the iterators [beg, end).*
- void [resize](#) ([size\\_type](#) num)  
*Resizes the [String](#) either by inserting default utf32 code points to make it larger, or by truncating to make it smaller.*
- void [resize](#) ([size\\_type](#) num, utf32 code\_point)  
*Resizes the [String](#) either by inserting the given utf32 code point to make it larger, or by truncating to make it smaller.*
- [String](#) & [replace](#) ([size\\_type](#) idx, [size\\_type](#) len, const [String](#) &str)  
*Replace code points in the [String](#) with the specified [String](#) object.*
- [String](#) & [replace](#) ([iterator](#) iter\_beg, [iterator](#) iter\_end, const [String](#) &str)  
*Replace the code points in the range [beg, end) with the specified [String](#) object.*
- [String](#) & [replace](#) ([size\\_type](#) idx, [size\\_type](#) len, const [String](#) &str, [size\\_type](#) str\_idx, [size\\_type](#) str\_num)  
*Replace code points in the [String](#) with the specified [String](#) object.*

Replace code points in the *String* with a specified sub-string of a given *String* object.

- *String* & *replace* (*size\_type* idx, *size\_type* len, const std::string &std\_str)  
Replace code points in the *String* with the specified std::string object.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, const std::string &std\_str)  
Replace the code points in the range [beg, end) with the specified std::string object.
- *String* & *replace* (*size\_type* idx, *size\_type* len, const std::string &std\_str, *size\_type* str\_idx, *size\_type* str\_num)  
Replace code points in the *String* with a specified sub-string of a given std::string object.
- *String* & *replace* (*size\_type* idx, *size\_type* len, const utf8 \*utf8\_str)  
Replace code points in the *String* with the specified null-terminated utf8 encoded data.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, const utf8 \*utf8\_str)  
Replace the code points in the range [beg, end) with the specified null-terminated utf8 encoded data.
- *String* & *replace* (*size\_type* idx, *size\_type* len, const utf8 \*utf8\_str, *size\_type* str\_len)  
Replace code points in the *String* with the specified utf8 encoded data.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, const utf8 \*utf8\_str, *size\_type* str\_len)  
Replace the code points in the range [beg, end) with the specified null-terminated utf8 encoded data.
- *String* & *replace* (*size\_type* idx, *size\_type* len, *size\_type* num, utf32 code\_point)  
Replaces a specified range of code points with occurrences of a given code point.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, *size\_type* num, utf32 code\_point)  
Replace the code points in the range [beg, end) with occurrences of a given code point.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, const\_iterator iter\_newBeg, const\_iterator iter\_newEnd)  
Replace the code points in the range [beg, end) with code points from the range [newBeg, newEnd).
- *String* & *replace* (*size\_type* idx, *size\_type* len, const char \*cstr)  
Replace code points in the *String* with the specified c-string.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, const char \*cstr)  
Replace the code points in the range [beg, end) with the specified c-string.
- *String* & *replace* (*size\_type* idx, *size\_type* len, const char \*chars, *size\_type* chars\_len)  
Replace code points in the *String* with chars from the given char array.
- *String* & *replace* (*iterator* iter\_beg, *iterator* iter\_end, const char \*chars, *size\_type* chars\_len)  
Replace the code points in the range [beg, end) with chars from the given char array.
- *size\_type* *find* (utf32 code\_point, *size\_type* idx=0) const  
Search forwards for a given code point.
- *size\_type* *rfind* (utf32 code\_point, *size\_type* idx=npos) const



*Search backwards for a given code point.*

- `size_type find` (const `String` &str, `size_type` idx=0) const  
*Search forwards for a sub-string.*
- `size_type rfind` (const `String` &str, `size_type` idx=npos) const  
*Search backwards for a sub-string.*
- `size_type find` (const std::string &std\_str, `size_type` idx=0) const  
*Search forwards for a sub-string.*
- `size_type rfind` (const std::string &std\_str, `size_type` idx=npos) const  
*Search backwards for a sub-string.*
- `size_type find` (const utf8 \*utf8\_str, `size_type` idx=0) const  
*Search forwards for a sub-string.*
- `size_type rfind` (const utf8 \*utf8\_str, `size_type` idx=npos) const  
*Search backwards for a sub-string.*
- `size_type find` (const utf8 \*utf8\_str, `size_type` idx, `size_type` str\_len) const  
*Search forwards for a sub-string.*
- `size_type rfind` (const utf8 \*utf8\_str, `size_type` idx, `size_type` str\_len) const  
*Search backwards for a sub-string.*
- `size_type find` (const char \*cstr, `size_type` idx=0) const  
*Search forwards for a sub-string.*
- `size_type rfind` (const char \*cstr, `size_type` idx=npos) const  
*Search backwards for a sub-string.*
- `size_type find` (const char \*chars, `size_type` idx, `size_type` chars\_len) const  
*Search forwards for a sub-string.*
- `size_type rfind` (const char \*chars, `size_type` idx, `size_type` chars\_len) const  
*Search backwards for a sub-string.*
- `size_type find_first_of` (const `String` &str, `size_type` idx=0) const  
*Find the first occurrence of one of a set of code points.*
- `size_type find_first_not_of` (const `String` &str, `size_type` idx=0) const  
*Find the first code point that is not one of a set of code points.*
- `size_type find_first_of` (const std::string &std\_str, `size_type` idx=0) const  
*Find the first occurrence of one of a set of code points.*
- `size_type find_first_not_of` (const std::string &std\_str, `size_type` idx=0) const  
*Find the first code point that is not one of a set of code points.*

- `size_type find_first_of` (const utf8 \*utf8\_str, `size_type` idx=0) const  
*Find the first occurrence of one of a set of code points.*
- `size_type find_first_not_of` (const utf8 \*utf8\_str, `size_type` idx=0) const  
*Find the first code point that is not one of a set of code points.*
- `size_type find_first_of` (const utf8 \*utf8\_str, `size_type` idx, `size_type` str\_len) const  
*Find the first occurrence of one of a set of code points.*
- `size_type find_first_not_of` (const utf8 \*utf8\_str, `size_type` idx, `size_type` str\_len) const  
*Find the first code point that is not one of a set of code points.*
- `size_type find_first_of` (utf32 code\_point, `size_type` idx=0) const  
*Search forwards for a given code point.*
- `size_type find_first_not_of` (utf32 code\_point, `size_type` idx=0) const  
*Search forwards for the first code point that does not match a given code point.*
- `size_type find_first_of` (const char \*cstr, `size_type` idx=0) const  
*Find the first occurrence of one of a set of chars.*
- `size_type find_first_not_of` (const char \*cstr, `size_type` idx=0) const  
*Find the first code point that is not one of a set of chars.*
- `size_type find_first_of` (const char \*chars, `size_type` idx, `size_type` chars\_len) const  
*Find the first occurrence of one of a set of chars.*
- `size_type find_first_not_of` (const char \*chars, `size_type` idx, `size_type` chars\_len) const  
*Find the first code point that is not one of a set of chars.*
- `size_type find_last_of` (const String &str, `size_type` idx=npos) const  
*Find the last occurrence of one of a set of code points.*
- `size_type find_last_not_of` (const String &str, `size_type` idx=npos) const  
*Find the last code point that is not one of a set of code points.*
- `size_type find_last_of` (const std::string &std\_str, `size_type` idx=npos) const  
*Find the last occurrence of one of a set of code points.*
- `size_type find_last_not_of` (const std::string &std\_str, `size_type` idx=npos) const  
*Find the last code point that is not one of a set of code points.*
- `size_type find_last_of` (const utf8 \*utf8\_str, `size_type` idx=npos) const  
*Find the last occurrence of one of a set of code points.*
- `size_type find_last_not_of` (const utf8 \*utf8\_str, `size_type` idx=npos) const  
*Find the last code point that is not one of a set of code points.*
- `size_type find_last_of` (const utf8 \*utf8\_str, `size_type` idx, `size_type` str\_len) const  
*Find the last occurrence of one of a set of code points.*

- `size_type find_last_not_of` (const utf8 \*utf8\_str, `size_type` idx, `size_type` str\_len) const  
*Find the last code point that is not one of a set of code points.*
- `size_type find_last_of` (utf32 code\_point, `size_type` idx=npos) const  
*Search for last occurrence of a given code point.*
- `size_type find_last_not_of` (utf32 code\_point, `size_type` idx=npos) const  
*Search for the last code point that does not match a given code point.*
- `size_type find_last_of` (const char \*cstr, `size_type` idx=npos) const  
*Find the last occurrence of one of a set of chars.*
- `size_type find_last_not_of` (const char \*cstr, `size_type` idx=npos) const  
*Find the last code point that is not one of a set of chars.*
- `size_type find_last_of` (const char \*chars, `size_type` idx, `size_type` chars\_len) const  
*Find the last occurrence of one of a set of chars.*
- `size_type find_last_not_of` (const char \*chars, `size_type` idx, `size_type` chars\_len) const  
*Find the last code point that is not one of a set of chars.*
- `String substr` (`size_type` idx=0, `size_type` len=npos) const  
*Returns a substring of this [String](#).*
- `iterator begin` (void)  
*Return a forwards [iterator](#) that describes the beginning of the [String](#).*
- `const_iterator begin` (void) const  
*Return a constant forwards [iterator](#) that describes the beginning of the [String](#).*
- `iterator end` (void)  
*Return a forwards [iterator](#) that describes the end of the [String](#).*
- `const_iterator end` (void) const  
*Return a constant forwards [iterator](#) that describes the end of the [String](#).*
- `reverse_iterator rbegin` (void)  
*Return a reverse [iterator](#) that describes the beginning of the [String](#).*
- `const_reverse_iterator rbegin` (void) const  
*Return a constant reverse [iterator](#) that describes the beginning of the [String](#).*
- `reverse_iterator rend` (void)  
*Return a reverse [iterator](#) that describes the end of the [String](#).*
- `const_reverse_iterator rend` (void) const  
*Return a constant reverse [iterator](#) that describes the end of the [String](#).*

## Static Public Member Functions

- static `size_type max_size` (void)  
*Returns the maximum size of a `String`.*

## Static Public Attributes

- static const `size_type npos` = `(String::size_type)(-1)`  
*Value used to represent 'not found' conditions and 'all code points' etc.*

## Classes

- class `const_iterator`  
*Constant forward `iterator` class for `String` objects.*
- struct `FastLessCompare`  
*Functor that can be used as comparator in a `std::map` with `String` keys. It's faster than using the default, but the map will no longer be sorted alphabetically.*
- class `iterator`  
*Forward `iterator` class for `String` objects.*

### 6.282.1 Detailed Description

`String` class used within the GUI system.

For the most part, this class can replace `std::string` in basic usage. However, currently `String` does not use the current locale, and also comparisons do not take into account the Unicode data tables, so are not 'correct' as such.

### 6.282.2 Constructor & Destructor Documentation

#### 6.282.2.1 `CEGUI::String::String (const String & str)` `[inline]`

Copy constructor - Creates a new string with the same value as *str*.

#### Parameters:

*str* `String` object used to initialise the newly created string

#### Returns:

Nothing

### 6.282.2.2 CEGUI::String::String (const String & *str*, size\_type *str\_idx*, size\_type *str\_num* = npos) [inline]

Constructs a new string initialised with code points from another [String](#) object.

#### Parameters:

*str* [String](#) object used to initialise the newly created string

*str\_idx* Starting code-point of *str* to be used when initialising the new [String](#)

*str\_num* Maximum number of code points from *str* that are to be assigned to the new [String](#)

#### Returns:

Nothing

### 6.282.2.3 CEGUI::String::String (const std::string & *std\_str*) [inline]

Constructs a new string and initialises it using the std::string *std\_str*.

#### Parameters:

*std\_str* The std::string object that is to be used to initialise the new [String](#) object.

#### Note:

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

#### Returns:

Nothing

#### Exceptions:

*std::length\_error* Thrown if resulting [String](#) object would be too big.

### 6.282.2.4 CEGUI::String::String (const std::string & *std\_str*, size\_type *str\_idx*, size\_type *str\_num* = npos) [inline]

Constructs a new string initialised with characters from the given std::string object.

#### Parameters:

*std\_str* std::string object used to initialise the newly created string

*str\_idx* Starting character of *std\_str* to be used when initialising the new [String](#)

#### Note:

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

#### Parameters:

*str\_num* Maximum number of characters from *std\_str* that are to be assigned to the new [String](#)

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

**6.282.2.5 CEGUI::String::String (const utf8 \* utf8\_str) [inline]**

Constructs a new [String](#) object and initialise it using the provided utf8 encoded string buffer.

**Parameters:**

*utf8\_str* Pointer to a buffer containing a null-terminated Unicode string encoded as utf8 data.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

**6.282.2.6 CEGUI::String::String (const utf8 \* utf8\_str, size\_type chars\_len) [inline]**

Constructs a new [String](#) object and initialise it using the provided utf8 encoded string buffer.

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*utf8\_str* Pointer to a buffer containing Unicode string data encoded as utf8.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*chars\_len* Length of the provided utf8 string in code units (not code-points).

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

**6.282.2.7 CEGUI::String::String (size\_type num, utf32 code\_point) [inline]**

Constructs a new [String](#) that is initialised with the specified code point.

**Parameters:**

*num* The number of times *code\_point* is to be put into new [String](#) object

*code\_point* The Unicode code point to be used when initialising the [String](#) object

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

**6.282.2.8 CEGUI::String::String (const\_iterator iter\_beg, const\_iterator iter\_end) [inline]**

Construct a new string object and initialise it with code-points from the range [beg, end).

**Parameters:**

*beg* Iterator describing the start of the data to be used when initialising the [String](#) object

*end* Iterator describing the (exclusive) end of the data to be used when initialising the [String](#) object

**Returns:**

Nothing

**6.282.2.9 CEGUI::String::String (const char \* cstr) [inline]**

Constructs a new [String](#) object and initialise it using the provided c-string.

**Parameters:**

*c\_str* Pointer to a c-string.

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

**6.282.2.10 CEGUI::String::String (const char \* chars, size\_type chars\_len) [inline]**

Constructs a new [String](#) object and initialise it using characters from the provided char array.

**Parameters:**

*chars* char array.

*chars\_len* Number of chars from the array to be used.

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

### 6.282.3 Member Function Documentation

#### 6.282.3.1 `size_type CEGUI::String::size (void) const` [inline]

Returns the size of the [String](#) in code points.

**Returns:**

Number of code points currently in the [String](#)

#### 6.282.3.2 `size_type CEGUI::String::length (void) const` [inline]

Returns the size of the [String](#) in code points.

**Returns:**

Number of code points currently in the [String](#)

#### 6.282.3.3 `bool CEGUI::String::empty (void) const` [inline]

Returns true if the [String](#) is empty.

**Returns:**

true if the [String](#) is empty, else false.

#### 6.282.3.4 `static size_type CEGUI::String::max_size (void)` [inline, static]

Returns the maximum size of a [String](#).

Any operation that would result in a [String](#) that is larger than this value will throw the `std::length_error` exception.

**Returns:**

The maximum number of code points that a string can contain



**6.282.3.5 size\_type CEGUI::String::capacity (void) const** [inline]

Return the number of code points that the [String](#) could hold before a re-allocation would be required.

**Returns:**

[Size](#) of the current reserve buffer. This is the maximum number of code points the [String](#) could hold before a buffer re-allocation would be required

**6.282.3.6 void CEGUI::String::reserve (size\_type num = 0)** [inline]

Specifies the amount of reserve capacity to allocate.

**Parameters:**

*num* The number of code points to allocate space for. If *num* is larger than the current reserve, then a re-allocation will occur. If *num* is smaller than the current reserve (but not 0) the buffer may be shrunk to the larger of the specified number, or the current [String](#) size (operation is currently not implemented). If *num* is 0, then the buffer is re-allocated to fit the current [String](#) size.

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) object would be too big.

**6.282.3.7 int CEGUI::String::compare (const String & str) const** [inline]

Compares this [String](#) with the [String](#) 'str'.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

*str* The [String](#) object that is to be compared with this [String](#).

**Returns:**

- 0 if the [String](#) objects are equal
- <0 if this [String](#) is lexicographically smaller than *str*
- >0 if this [String](#) is lexicographically greater than *str*

**6.282.3.8 int CEGUI::String::compare (size\_type idx, size\_type len, const String & str, size\_type str\_idx = 0, size\_type str\_len = npos) const** [inline]

Compares code points from this [String](#) with code points from the [String](#) 'str'.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

*idx* Index of the first code point from this [String](#) to consider.  
*len* Maximum number of code points from this [String](#) to consider.  
*str* The [String](#) object that is to be compared with this [String](#).  
*str\_idx* Index of the first code point from [String](#) *str* to consider.  
*str\_len* Maximum number of code points from [String](#) *str* to consider

**Returns:**

- 0 if the specified sub-strings are equal
- <0 if specified sub-strings are lexicographically smaller than *str*
- >0 if specified sub-strings are lexicographically greater than *str*

**Exceptions:**

*std::out\_of\_range* Thrown if either *idx* or *str\_idx* are invalid.

### 6.282.3.9 `int CEGUI::String::compare (const std::string & std_str) const` `[inline]`

Compares this [String](#) with the `std::string` '*std\_str*'.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

*std\_str* The `std::string` object that is to be compared with this [String](#).

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Returns:**

- 0 if the string objects are equal
- <0 if this string is lexicographically smaller than *std\_str*
- >0 if this string is lexicographically greater than *std\_str*

### 6.282.3.10 `int CEGUI::String::compare (size_type idx, size_type len, const std::string & std_str, size_type str_idx = 0, size_type str_len = npos) const` `[inline]`

Compares code points from this [String](#) with code points from the `std::string` '*std\_str*'.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

- idx* Index of the first code point from this [String](#) to consider.
- len* Maximum number of code points from this [String](#) to consider.
- std\_str* The std::string object that is to be compared with this [String](#).

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Parameters:**

- str\_idx* Index of the first character from std::string *std\_str* to consider.
- str\_len* Maximum number of characters from std::string *std\_str* to consider

**Returns:**

- 0 if the specified sub-strings are equal
- <0 if specified sub-strings are lexicographically smaller than *std\_str*
- >0 if specified sub-strings are lexicographically greater than *std\_str*

**Exceptions:**

- std::out\_of\_range* Thrown if either *idx* or *str\_idx* are invalid.

**6.282.3.11 int CEGUI::String::compare (const utf8 \* utf8\_str) const** [inline]

Compares this [String](#) with the null-terminated utf8 encoded 'utf8\_str'.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

- utf8\_str* The buffer containing valid Unicode data encoded as utf8 that is to be compared with this [String](#).

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

- 0 if the strings are equal
- <0 if this string is lexicographically smaller than *utf8\_str*
- >0 if this string is lexicographically greater than *utf8\_str*

### 6.282.3.12 `int CEGUI::String::compare (size_type idx, size_type len, const utf8 * utf8_str) const` `[inline]`

Compares code points from this [String](#) with the null-terminated utf8 encoded 'utf8\_str'.

#### Note:

This does currently not properly consider Unicode and / or the system locale.

#### Parameters:

*idx* Index of the first code point from this [String](#) to consider.

*len* Maximum number of code points from this [String](#) to consider.

*utf8\_str* The buffer containing valid Unicode data encoded as utf8 that is to be compared with this [String](#).

#### Note:

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

#### Returns:

- 0 if the specified sub-strings are equal
- <0 if specified sub-strings are lexicographically smaller than *utf8\_str*
- >0 if specified sub-strings are lexicographically greater than *utf8\_str*

#### Exceptions:

*std::out\_of\_range* Thrown if *idx* is invalid.

### 6.282.3.13 `int CEGUI::String::compare (size_type idx, size_type len, const utf8 * utf8_str, size_type str_cplen) const` `[inline]`

Compares code points from this [String](#) with the utf8 encoded data in buffer 'utf8\_str'.

#### Note:

This does currently not properly consider Unicode and / or the system locale.

#### Parameters:

*idx* Index of the first code point from this [String](#) to consider.

*len* Maximum number of code points from this [String](#) to consider.

*utf8\_str* The buffer containing valid Unicode data encoded as utf8 that is to be compared with this [String](#).

#### Note:

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*str\_cplen* The number of encoded code points in the buffer *utf8\_str* (this is not the same as the number of code units).

**Returns:**

- 0 if the specified sub-strings are equal
- <0 if specified sub-strings are lexicographically smaller than *utf8\_str*
- >0 if specified sub-strings are lexicographically greater than *utf8\_str*

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid.

*std::length\_error* Thrown if *str\_cplen* is set to npos.

**6.282.3.14 int CEGUI::String::compare (const char \* *cstr*) const** [inline]

Compares this [String](#) with the given c-string.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

*c\_str* The c-string that is to compared with this [String](#).

**Returns:**

- 0 if the strings are equal
- <0 if this string is lexicographically smaller than *c\_str*
- >0 if this string is lexicographically greater than *c\_str*

**6.282.3.15 int CEGUI::String::compare (size\_type *idx*, size\_type *len*, const char \* *cstr*) const** [inline]

Compares code points from this [String](#) with the given c-string.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

*idx* Index of the first code point from this [String](#) to consider.

*len* Maximum number of code points from this [String](#) to consider.

*c\_str* The c-string that is to compared with this [String](#).

**Returns:**

- 0 if the specified sub-strings are equal

- <0 if specified sub-strings are lexicographically smaller than *c\_str*
- >0 if specified sub-strings are lexicographically greater than *c\_str*

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid.

### 6.282.3.16 `int CEGUI::String::compare (size_type idx, size_type len, const char * chars, size_type chars_len) const` [inline]

Compares code points from this [String](#) with chars in the given char array.

**Note:**

This does currently not properly consider Unicode and / or the system locale.

**Parameters:**

*idx* Index of the first code point from this [String](#) to consider.  
*len* Maximum number of code points from this [String](#) to consider.  
*chars* The array containing the chars that are to compared with this [String](#).  
*chars\_len* The number of chars in the array.

**Returns:**

- 0 if the specified sub-strings are equal
- <0 if specified sub-strings are lexicographically smaller than *chars*
- >0 if specified sub-strings are lexicographically greater than *chars*

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid.  
*std::length\_error* Thrown if *chars\_len* is set to npos.

### 6.282.3.17 `reference CEGUI::String::operator[] (size_type idx)` [inline]

Returns the code point at the given index.

**Parameters:**

*idx* Zero based index of the code point to be returned.

**Note:**

- For constant strings [length\(\)](#)/[size\(\)](#) provide a valid index and will access the default utf32 value.
- For non-constant strings [length\(\)](#)/[size\(\)](#) is an invalid index, and accessing (especially writing) this index could cause string corruption.

**Returns:**

The utf32 code point at the given index within the [String](#).

**6.282.3.18 value\_type CEGUI::String::operator[] (size\_type *idx*) const** [inline]

Returns the code point at the given index.

**Parameters:**

*idx* Zero based index of the code point to be returned.

**Note:**

- For constant strings [length\(\)/size\(\)](#) provide a valid index and will access the default utf32 value.
- For non-constant strings [length\(\)/size\(\)](#) is an invalid index, and accessing (especially writing) this index could cause string corruption.

**Returns:**

The utf32 code point at the given index within the [String](#).

**6.282.3.19 reference CEGUI::String::at (size\_type *idx*)** [inline]

Returns the code point at the given index.

**Parameters:**

*idx* Zero based index of the code point to be returned.

**Returns:**

The utf32 code point at the given index within the [String](#).

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is  $\geq$  [length\(\)](#).

**6.282.3.20 const\_reference CEGUI::String::at (size\_type *idx*) const** [inline]

Returns the code point at the given index.

**Parameters:**

*idx* Zero based index of the code point to be returned.

**Returns:**

The utf32 code point at the given index within the [String](#).

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is  $\geq$  [length\(\)](#).

**6.282.3.21** `const char* CEGUI::String::c_str (void) const` `[inline]`

Returns contents of the [String](#) as a null terminated string of utf8 encoded data.

**Returns:**

Pointer to a char buffer containing the contents of the [String](#) encoded as null-terminated utf8 data.

**Note:**

The buffer returned from this function is owned by the [String](#) object.

Any function that modifies the [String](#) data will invalidate the buffer returned by this call.

**6.282.3.22** `const utf8* CEGUI::String::data (void) const` `[inline]`

Returns contents of the [String](#) as utf8 encoded data.

**Returns:**

Pointer to a buffer containing the contents of the [String](#) encoded utf8 data.

**Note:**

The buffer returned from this function is owned by the [String](#) object.

Any function that modifies the [String](#) data will invalidate the buffer returned by this call.

**6.282.3.23** `size_type CEGUI::String::copy (utf8 * buf, size_type len = npos, size_type idx = 0) const` `[inline]`

Copies an area of the [String](#) into the provided buffer as encoded utf8 data.

**Parameters:**

*buf* Pointer to a buffer that is to receive the encoded data (this must be big enough to hold the encoded data)

*len* Maximum number of code points from the [String](#) that should be encoded into the buffer

*idx* Index of the first code point to be encoded into the buffer

**Returns:**

The number of utf8 encoded code units transferred to the buffer.

**Note:**

A code unit does not equal a code point. A utf32 code point, when encoded as utf8, can occupy between 1 and 4 code units.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* was invalid for this [String](#).



**6.282.3.24** `size_type CEGUI::String::utf8_stream_len (size_type num = npos, size_type idx = 0) const [inline]`

Return the number of utf8 code units required to hold an area of the [String](#) when encoded as utf8 data.

**Parameters:**

*num* Maximum number of code points to consider when calculating utf8 encoded size.

*idx* Index of the first code point to consider when calculating the utf8 encoded size

**Returns:**

The number of utf8 code units (bytes) required to hold the specified sub-string when encoded as utf8 data.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* was invalid for this [String](#).

**6.282.3.25** `String& CEGUI::String::operator= (const String & str) [inline]`

Assign the value of [String](#) *str* to this [String](#).

**Parameters:**

*str* [String](#) object containing the string value to be assigned.

**Returns:**

This [String](#) after the assignment has happened

**6.282.3.26** `String& CEGUI::String::assign (const String & str, size_type str_idx = 0, size_type str_num = npos) [inline]`

Assign a sub-string of [String](#) *str* to this [String](#).

**Parameters:**

*str* [String](#) object containing the string data to be assigned.

*str\_idx* Index of the first code point in *str* that is to be assigned

*str\_num* Maximum number of code points from *str* that are to be assigned

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::out\_of\_range* Thrown if *str\_idx* is invalid for *str*

**6.282.3.27 String& CEGUI::String::operator= (const std::string & *std\_str*) [inline]**

Assign the value of std::string *std\_str* to this [String](#).

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*std\_str* std::string object containing the string value to be assigned.

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.28 String& CEGUI::String::assign (const std::string & *std\_str*, size\_type *str\_idx* = 0, size\_type *str\_num* = npos) [inline]**

Assign a sub-string of std::string *std\_str* to this [String](#).

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*std\_str* std::string object containing the string value to be assigned.

*str\_idx* Index of the first character of *std\_str* to be assigned

*str\_num* Maximum number of characters from *std\_str* to be assigned

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::out\_of\_range* Thrown if *str\_idx* is invalid for *std\_str*

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.29 String& CEGUI::String::operator= (const utf8 \* *utf8\_str*) [inline]**

Assign to this [String](#) the string value represented by the given null-terminated utf8 encoded data.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*utf8\_str* Buffer containing valid null-terminated utf8 encoded data

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.30 String& CEGUI::String::assign (const utf8 \* utf8\_str) [inline]**

Assign to this [String](#) the string value represented by the given null-terminated utf8 encoded data.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*utf8\_str* Buffer containing valid null-terminated utf8 encoded data

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.31 String& CEGUI::String::assign (const utf8 \* utf8\_str, size\_type str\_num) [inline]**

Assign to this [String](#) the string value represented by the given utf8 encoded data.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*utf8\_str* Buffer containing valid utf8 encoded data

*str\_num* Number of code units (not code points) in the buffer pointed to by *utf8\_str*

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large, or if *str\_num* is 'npos'.

**6.282.3.32 String& CEGUI::String::operator= (utf32 *code\_point*)** [inline]

Assigns the specified utf32 code point to this [String](#). Result is always a [String](#) 1 code point in length.

**Parameters:**

*code\_point* Valid utf32 Unicode code point to be assigned to the string

**Returns:**

This [String](#) after assignment

**6.282.3.33 String& CEGUI::String::assign (size\_type *num*, utf32 *code\_point*)** [inline]

Assigns the specified code point repeatedly to the [String](#).

**Parameters:**

*num* The number of times to assign the code point

*code\_point* Valid utf32 Unicode code point to be assigned to the string

**Returns:**

This [String](#) after assignment.

**Exceptions:**

*std::length\_error* Thrown if *num* was 'npos'

**6.282.3.34 String& CEGUI::String::operator= (const char \* *cstr*)** [inline]

Assign to this [String](#) the given C-string.

**Parameters:**

*c\_str* Pointer to a valid C style string.

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.35 String& CEGUI::String::assign (const char \* *cstr*)** [inline]

Assign to this [String](#) the given C-string.

**Parameters:**

*c\_str* Pointer to a valid C style string.

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.36 String& CEGUI::String::assign (const char \* *chars*, size\_type *chars\_len*)** `[inline]`

Assign to this [String](#) a number of chars from a char array.

**Parameters:**

*chars* char array.

*chars\_len* Number of chars to be assigned.

**Returns:**

This [String](#) after the assignment has happened

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.37 void CEGUI::String::swap (String & *str*)** `[inline]`

Swaps the value of this [String](#) with the given [String](#) *str*.

**Parameters:**

*str* [String](#) object whos value is to be swapped with this [String](#).

**Returns:**

Nothing

**6.282.3.38 String& CEGUI::String::operator+= (const String & *str*)** `[inline]`

Appends the [String](#) *str*.

**Parameters:**

*str* [String](#) object that is to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.39 String& CEGUI::String::append (const String & *str*, size\_type *str\_idx* = 0, size\_type *str\_num* = npos) [inline]**

Appends a sub-string of the [String](#) *str*.

**Parameters:**

*str* [String](#) object containing data to be appended  
*str\_idx* Index of the first code point to be appended  
*str\_num* Maximum number of code points to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::out\_of\_range* Thrown if *str\_idx* is invalid for *str*.  
*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.40 String& CEGUI::String::operator+= (const std::string & *std\_str*) [inline]**

Appends the std::string *std\_str*.

**Parameters:**

*std\_str* std::string object that is to be appended

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.41 String& CEGUI::String::append (const std::string & *std\_str*, size\_type *str\_idx* = 0, size\_type *str\_num* = npos) [inline]**

Appends a sub-string of the std::string *std\_str*.

**Parameters:**

*std\_str* std::string object containing data to be appended

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*str\_idx* Index of the first character to be appended  
*str\_num* Maximum number of characters to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::out\_of\_range* Thrown if *str\_idx* is invalid for *std\_str*.  
*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.42 String& CEGUI::String::operator+=(const utf8 \* utf8\_str) [inline]**

Appends to the [String](#) the null-terminated utf8 encoded data in the buffer utf8\_str.

**Parameters:**

*utf8\_str* buffer holding the null-terminated utf8 encoded data that is to be appended

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.43 String& CEGUI::String::append(const utf8 \* utf8\_str) [inline]**

Appends to the [String](#) the null-terminated utf8 encoded data in the buffer utf8\_str.

**Parameters:**

*utf8\_str* Buffer holding the null-terminated utf8 encoded data that is to be appended

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.44 String& CEGUI::String::append (const utf8 \* *utf8\_str*, size\_type *len*)** [inline]

Appends to the [String](#) the utf8 encoded data in the buffer *utf8\_str*.

**Parameters:**

*utf8\_str* Buffer holding the utf8 encoded data that is to be appended

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*len* Number of code units (not code points) in the buffer to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large, or if *len* was 'npos'

**6.282.3.45 String& CEGUI::String::operator+= (utf32 *code\_point*)** [inline]

Appends a single code point to the string.

**Parameters:**

*code\_point* utf32 Unicode code point that is to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too long.

**6.282.3.46 String& CEGUI::String::append (size\_type *num*, utf32 *code\_point*)** [inline]

Appends a single code point multiple times to the string.

**Parameters:**

*num* Number of copies of the code point to be appended

*code\_point* utf32 Unicode code point that is to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too long, or if *num* was 'npos'.



**6.282.3.47** void CEGUI::String::push\_back (utf32 *code\_point*) [inline]

Appends a single code point to the string.

**Parameters:**

*code\_point* utf32 Unicode code point that is to be appended

**Returns:**

Nothing

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too long.

**6.282.3.48** String& CEGUI::String::append (const\_iterator *iter\_beg*, const\_iterator *iter\_end*) [inline]

Appends the code points in the reange [beg, end).

**Parameters:**

*beg* Iterator describing the start of the range to be appended

*end* Iterator describing the (exclusive) end of the range to be appended.

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if the resulting string would be too large.

**6.282.3.49** String& CEGUI::String::operator+= (const char \* *cstr*) [inline]

Appends to the [String](#) the given c-string.

**Parameters:**

*c\_str* c-string that is to be appended.

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.50 String& CEGUI::String::append (const char \* *cstr*)** [inline]

Appends to the [String](#) the given c-string.

**Parameters:**

*c\_str* c-string that is to be appended.

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.51 String& CEGUI::String::append (const char \* *chars*, size\_type *chars\_len*)** [inline]

Appends to the [String](#) chars from the given char array.

**Parameters:**

*chars* char array holding the chars that are to be appended

*chars\_len* Number of chars to be appended

**Returns:**

This [String](#) after the append operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large, or if *chars\_len* was 'npos'

**6.282.3.52 String& CEGUI::String::insert (size\_type *idx*, const String & *str*)** [inline]

Inserts the given [String](#) object at the specified position.

**Parameters:**

*idx* Index where the string is to be inserted.

*str* [String](#) object that is to be inserted.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.53 String& CEGUI::String::insert (size\_type *idx*, const String & *str*, size\_type *str\_idx*, size\_type *str\_num*)** [inline]

Inserts a sub-string of the given [String](#) object at the specified position.

**Parameters:**

- idx* Index where the string is to be inserted.
- str* [String](#) object containing data to be inserted.
- str\_idx* Index of the first code point from *str* to be inserted.
- str\_num* Maximum number of code points from *str* to be inserted.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

- std::out\_of\_range* Thrown if *idx* or *str\_idx* are out of range.
- std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.54 String& CEGUI::String::insert (size\_type *idx*, const std::string & *std\_str*)** [inline]

Inserts the given std::string object at the specified position.

**Parameters:**

- idx* Index where the std::string is to be inserted.
- std\_str* std::string object that is to be inserted.

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

- std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).
- std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.55 String& CEGUI::String::insert (size\_type *idx*, const std::string & *std\_str*, size\_type *str\_idx*, size\_type *str\_num*)** [inline]

Inserts a sub-string of the given std::string object at the specified position.

**Parameters:**

- idx* Index where the string is to be inserted.

*std\_str* std::string object containing data to be inserted.

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*str\_idx* Index of the first character from *std\_str* to be inserted.

*str\_num* Maximum number of characters from *str* to be inserted.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* or *str\_idx* are out of range.

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.56 String& CEGUI::String::insert (size\_type idx, const utf8 \* utf8\_str) [inline]**

Inserts the given null-terminated utf8 encoded data at the specified position.

**Parameters:**

*idx* Index where the data is to be inserted.

*utf8\_str* Buffer containing the null-terminated utf8 encoded data that is to be inserted.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.57 String& CEGUI::String::insert (size\_type idx, const utf8 \* utf8\_str, size\_type len) [inline]**

Inserts the given utf8 encoded data at the specified position.

**Parameters:**

*idx* Index where the data is to be inserted.

*utf8\_str* Buffer containing the utf8 encoded data that is to be inserted.

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points `0x00..0x7f`. The use of extended ASCII characters (with values `>0x7f`) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*len* Length of the data to be inserted in utf8 code units (not code points)

**Returns:**

This [String](#) after the insert.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

*std::length\_error* Thrown if resulting [String](#) would be too large, or if *len* is 'npos'

### 6.282.3.58 `String& CEGUI::String::insert (size_type idx, size_type num, utf32 code_point)` [inline]

Inserts a code point multiple times into the [String](#).

**Parameters:**

*idx* Index where the code point(s) are to be inserted

*num* The number of times to insert the code point

*code\_point* The utf32 code point that is to be inserted

**Returns:**

This [String](#) after the insertion.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

*std::length\_error* Thrown if resulting [String](#) would be too large, or if *num* is 'npos'

### 6.282.3.59 `void CEGUI::String::insert (iterator pos, size_type num, utf32 code_point)` [inline]

Inserts a code point multiple times into the [String](#).

**Parameters:**

*pos* Iterator describing the position where the code point(s) are to be inserted

*num* The number of times to insert the code point

*code\_point* The utf32 code point that is to be inserted

**Returns:**

This [String](#) after the insertion.

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large, or if *num* is 'npos'

**6.282.3.60** `iterator CEGUI::String::insert (iterator pos, utf32 code_point)` `[inline]`

Inserts a single code point into the [String](#).

**Parameters:**

*pos* Iterator describing the position where the code point is to be inserted

*code\_point* The utf32 code point that is to be inserted

**Returns:**

This [String](#) after the insertion.

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.61** `void CEGUI::String::insert (iterator iter_pos, const_iterator iter_beg, const_iterator iter_end)` `[inline]`

Inserts code points specified by the range [*beg*, *end*).

**Parameters:**

*pos* Iterator describing the position where the data is to be inserted

*beg* Iterator describing the beginning of the range to be inserted

*end* Iterator describing the (exclusive) end of the range to be inserted.

**Returns:**

Nothing.

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would be too large.

**6.282.3.62** `String& CEGUI::String::insert (size_type idx, const char * cstr)` `[inline]`

Inserts the given c-string at the specified position.

**Parameters:**

*idx* Index where the c-string is to be inserted.

*c\_str* c-string that is to be inserted.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

*std::length\_error* Thrown if resulting [String](#) would be too large.

### 6.282.3.63 [String](#)& CEGUI::String::insert (size\_type *idx*, const char \* *chars*, size\_type *chars\_len*) [inline]

Inserts chars from the given char array at the specified position.

**Parameters:**

*idx* Index where the data is to be inserted.

*chars* char array containing the chars that are to be inserted.

*chars\_len* Length of the char array to be inserted.

**Returns:**

This [String](#) after the insert.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

*std::length\_error* Thrown if resulting [String](#) would be too large, or if *chars\_len* is 'npos'

### 6.282.3.64 void CEGUI::String::clear (void) [inline]

Removes all data from the [String](#).

**Returns:**

Nothing

### 6.282.3.65 [String](#)& CEGUI::String::erase (void) [inline]

Removes all data from the [String](#).

**Returns:**

The empty [String](#) (\*this)

**6.282.3.66 String& CEGUI::String::erase (size\_type *idx*)** `[inline]`

Erase a single code point from the string.

**Parameters:**

*idx* The index of the code point to be removed.

**Returns:**

This [String](#) after the erase operation

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.67 String& CEGUI::String::erase (size\_type *idx*, size\_type *len* = npos)** `[inline]`

Erase a range of code points.

**Parameters:**

*idx* Index of the first code point to be removed.

*len* Maximum number of code points to be removed.

**Returns:**

This [String](#) after the erase operation.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.68 String& CEGUI::String::erase (iterator *pos*)** `[inline]`

Erase the code point described by the given [iterator](#).

**Parameters:**

*pos* Iterator describing the code point to be erased

**Returns:**

This [String](#) after the erase operation.

**6.282.3.69 String& CEGUI::String::erase (iterator *iter\_beg*, iterator *iter\_end*)** `[inline]`

Erase a range of code points described by the iterators [*beg*, *end*).

**Parameters:**

*beg* Iterator describing the position of the beginning of the range to erase

*end* Iterator describing the position of the (exclusive) end of the range to erase

**Returns:**

This [String](#) after the erase operation.



**6.282.3.70 void CEGUI::String::resize (size\_type *num*) [inline]**

Resizes the [String](#) either by inserting default utf32 code points to make it larger, or by truncating to make it smaller.

**Parameters:**

*num* The length, in code points, that the [String](#) is to be made.

**Returns:**

Nothing.

**Exceptions:**

*std::length\_error* Thrown if the [String](#) would be too large.

**6.282.3.71 void CEGUI::String::resize (size\_type *num*, utf32 code\_point) [inline]**

Resizes the [String](#) either by inserting the given utf32 code point to make it larger, or by truncating to make it smaller.

**Parameters:**

*num* The length, in code points, that the [String](#) is to be made.

*code\_point* The utf32 code point that should be used when making the [String](#) larger

**Returns:**

Nothing.

**Exceptions:**

*std::length\_error* Thrown if the [String](#) would be too large.

**6.282.3.72 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const String & *str*) [inline]**

Replace code points in the [String](#) with the specified [String](#) object.

**Parameters:**

*idx* Index of the first code point to be replaced

*len* Maximum number of code points to be replaced (if this is 0, operation is an insert at position *idx*)

*str* The [String](#) object that is to replace the specified code points

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.73 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const String & *str*) [inline]**

Replace the code points in the range [beg, end) with the specified [String](#) object.

**Note:**

If *beg* == *end*, the operation is a insert at [iterator](#) position *beg*

**Parameters:**

*beg* Iterator describing the start of the range to be replaced

*end* Iterator describing the (exclusive) end of the range to be replaced.

*str* The [String](#) object that is to replace the specified range of code points

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.74 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const String & *str*, size\_type *str\_idx*, size\_type *str\_num*) [inline]**

Replace code points in the [String](#) with a specified sub-string of a given [String](#) object.

**Parameters:**

*idx* Index of the first code point to be replaced

*len* Maximum number of code points to be replaced. If this is 0, the operation is an insert at position *idx*.

*str* [String](#) object containing the data that will replace the specified range of code points

*str\_idx* Index of the first code point of *str* that is to replace the specified code point range

*str\_num* Maximum number of code points of *str* that are to replace the specified code point range

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::out\_of\_range* Thrown if either *idx*, or *str\_idx* are invalid

*std::length\_error* Thrown if the resulting [String](#) would have been too large.

**6.282.3.75 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const std::string & *std\_str*) [inline]**

Replace code points in the [String](#) with the specified std::string object.

**Parameters:**

*idx* Index of the first code point to be replaced  
*len* Maximum number of code points to be replaced (if this is 0, operation is an insert at position *idx*)  
*std\_str* The std::string object that is to replace the specified code points

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)  
*std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.76 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const std::string & *std\_str*) [inline]**

Replace the code points in the range [beg, end) with the specified std::string object.

**Note:**

If *beg* == *end*, the operation is a insert at [iterator](#) position *beg*

**Parameters:**

*beg* Iterator describing the start of the range to be replaced  
*end* Iterator describing the (exclusive) end of the range to be replaced.  
*std\_str* The std::string object that is to replace the specified range of code points

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.77 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const std::string & *std\_str*, size\_type *str\_idx*, size\_type *str\_num*) [inline]**

Replace code points in the [String](#) with a specified sub-string of a given std::string object.

**Parameters:**

*idx* Index of the first code point to be replaced  
*len* Maximum number of code points to be replaced. If this is 0, the operation is an insert at position *idx*.  
*std\_str* std::string object containing the data that will replace the specified range of code points

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Parameters:**

*str\_idx* Index of the first code point of *std\_str* that is to replace the specified code point range  
*str\_num* Maximum number of code points of *std\_str* that are to replace the specified code point range

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::out\_of\_range* Thrown if either *idx*, or *str\_idx* are invalid  
*std::length\_error* Thrown if the resulting [String](#) would have been too large.

### 6.282.3.78 **String& CEGUI::String::replace (size\_type idx, size\_type len, const utf8 \* utf8\_str)** [inline]

Replace code points in the [String](#) with the specified null-terminated utf8 encoded data.

**Parameters:**

*idx* Index of the first code point to be replaced  
*len* Maximum number of code points to be replaced (if this is 0, operation is an insert at position *idx*)  
*utf8\_str* Buffer containing the null-terminated utf8 encoded data that is to replace the specified code points

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)  
*std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.79 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const utf8 \* *utf8\_str*) [inline]**

Replace the code points in the range [*beg*, *end*) with the specified null-terminated utf8 encoded data.

**Note:**

If *beg* == *end*, the operation is a insert at [iterator](#) position *beg*

**Parameters:**

*beg* Iterator describing the start of the range to be replaced

*end* Iterator describing the (exclusive) end of the range to be replaced.

*utf8\_str* Buffer containing the null-terminated utf8 encoded data that is to replace the specified range of code points

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.80 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const utf8 \* *utf8\_str*, size\_type *str\_len*) [inline]**

Replace code points in the [String](#) with the specified utf8 encoded data.

**Parameters:**

*idx* Index of the first code point to be replaced

*len* Maximum number of code points to be replaced (if this is 0, operation is an insert at position *idx*)

*utf8\_str* Buffer containing the null-terminated utf8 encoded data that is to replace the specified code points

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*str\_len* Length of the utf8 encoded data in utf8 code units (not code points).

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)

*std::length\_error* Thrown if the resulting [String](#) would be too large, or if *str\_len* was 'npos'.

**6.282.3.81 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const utf8 \*  
utf8\_str, size\_type *str\_len*) [inline]**

Replace the code points in the range [beg, end) with the specified null-terminated utf8 encoded data.

**Note:**

If *beg* == *end*, the operation is a insert at [iterator](#) position *beg*

**Parameters:**

*beg* Iterator describing the start of the range to be replaced

*end* Iterator describing the (exclusive) end of the range to be replaced.

*utf8\_str* Buffer containing the null-terminated utf8 encoded data that is to replace the specified range of code points

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*str\_len* Length of the utf8 encoded data in utf8 code units (not code points).

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::length\_error* Thrown if the resulting [String](#) would be too large, or if *str\_len* was 'npos'.

**6.282.3.82 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, size\_type *num*, utf32  
code\_point) [inline]**

Replaces a specified range of code points with occurrences of a given code point.

**Parameters:**

*idx* Index of the first code point to be replaced

*len* Maximum number of code points to replace. If this is 0 the operation is an insert

*num* Number of occurrences of *code\_point* that are to replace the specified range of code points

*code\_point* Code point that is to be used when replacing the specified range of code points

**Returns:**

This [String](#) after the replace operation.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)

*std::length\_error* Thrown if resulting [String](#) would have been too long, or if *num* was 'npos'.

**6.282.3.83 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, size\_type *num*, utf32 *code\_point*) [inline]**

Replace the code points in the range [*beg*, *end*) with occurrences of a given code point.

**Note:**

If *beg* == *end*, the operation is an insert at [iterator](#) position *beg*

**Parameters:**

*beg* Iterator describing the start of the range to be replaced

*end* Iterator describing the (exclusive) end of the range to be replaced.

*num* Number of occurrences of *code\_point* that are to replace the specified range of code points

*code\_point* Code point that is to be used when replacing the specified range of code points

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

*std::length\_error* Thrown if resulting [String](#) would have been too long, or if *num* was 'npos'.

**6.282.3.84 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const\_iterator *iter\_newBeg*, const\_iterator *iter\_newEnd*) [inline]**

Replace the code points in the range [*beg*, *end*) with code points from the range [*newBeg*, *newEnd*).

**Note:**

If *beg* == *end*, the operation is an insert at [iterator](#) position *beg*

**Parameters:**

*beg* Iterator describing the start of the range to be replaced

*end* Iterator describing the (exclusive) end of the range to be replaced.

*newBeg* Iterator describing the beginning of the range to insert.

*newEnd* Iterator describing the (exclusive) end of the range to insert.

**Returns:**

This [String](#) after the insert operation.

**Exceptions:**

*std::length\_error* Thrown if the resulting string would be too long.

**6.282.3.85 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const char \* *cstr*)**  
[inline]

Replace code points in the [String](#) with the specified c-string.

**Parameters:**

- idx* Index of the first code point to be replaced
- len* Maximum number of code points to be replaced (if this is 0, operation is an insert at position *idx*)
- c\_str* c-string that is to replace the specified code points

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

- std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)
- std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.86 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const char \* *cstr*)**  
[inline]

Replace the code points in the range [beg, end) with the specified c-string.

**Note:**

If *beg* == *end*, the operation is a insert at [iterator](#) position *beg*

**Parameters:**

- beg* Iterator describing the start of the range to be replaced
- end* Iterator describing the (exclusive) end of the range to be replaced.
- c\_str* c-string that is to replace the specified range of code points

**Returns:**

This [String](#) after the replace operation

**Exceptions:**

- std::length\_error* Thrown if the resulting [String](#) would be too large.

**6.282.3.87 String& CEGUI::String::replace (size\_type *idx*, size\_type *len*, const char \* *chars*, size\_type *chars\_len*)** [inline]

Replace code points in the [String](#) with chars from the given char array.

**Parameters:**

- idx* Index of the first code point to be replaced
- len* Maximum number of code points to be replaced (if this is 0, operation is an insert at position *idx*)



*chars* char array containing the chars that are to replace the specified code points

*chars\_len* Number of chars in the char array.

#### Returns:

This [String](#) after the replace operation

#### Exceptions:

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#)

*std::length\_error* Thrown if the resulting [String](#) would be too large, or if *chars\_len* was 'npos'.

### 6.282.3.88 String& CEGUI::String::replace (iterator *iter\_beg*, iterator *iter\_end*, const char \* *chars*, size\_type *chars\_len*) [inline]

Replace the code points in the range [beg, end) with chars from the given char array.

#### Note:

If *beg* == *end*, the operation is a insert at [iterator](#) position *beg*

#### Parameters:

*beg* Iterator describing the start of the range to be replaced

*end* Iterator describing the (exclusive) end of the range to be replaced.

*chars* char array containing the chars that are to replace the specified range of code points

*chars\_len* Number of chars in the char array.

#### Returns:

This [String](#) after the replace operation

#### Exceptions:

*std::length\_error* Thrown if the resulting [String](#) would be too large, or if *chars\_len* was 'npos'.

### 6.282.3.89 size\_type CEGUI::String::find (utf32 *code\_point*, size\_type *idx* = 0) const [inline]

Search forwards for a given code point.

#### Parameters:

*code\_point* The utf32 code point to search for

*idx* Index of the code point where the search is to start.

#### Returns:

- Index of the first occurrence of *code\_point* travelling forwards from *idx*.
- npos if the code point could not be found

**6.282.3.90** `size_type CEGUI::String::rfind (utf32 code_point, size_type idx = npos) const`  
[inline]

Search backwards for a given code point.

**Parameters:**

*code\_point* The utf32 code point to search for  
*idx* Index of the code point where the search is to start.

**Returns:**

- Index of the first occurrence of *code\_point* travelling backwards from *idx*.
- npos if the code point could not be found

**6.282.3.91** `size_type CEGUI::String::find (const String & str, size_type idx = 0) const` [inline]

Search forwards for a sub-string.

**Parameters:**

*str* [String](#) object describing the sub-string to search for  
*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *str* travelling forwards from *idx*.
- npos if the sub-string could not be found

**6.282.3.92** `size_type CEGUI::String::rfind (const String & str, size_type idx = npos) const`  
[inline]

Search backwards for a sub-string.

**Parameters:**

*str* [String](#) object describing the sub-string to search for  
*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *str* travelling backwards from *idx*.
- npos if the sub-string could not be found

**6.282.3.93** `size_type CEGUI::String::find (const std::string & std_str, size_type idx = 0) const`  
[inline]

Search forwards for a sub-string.

**Parameters:**

*std\_str* std::string object describing the sub-string to search for

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Parameters:**

*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *std\_str* travelling forwards from *idx*.
- npos if the sub-string could not be found

**6.282.3.94** `size_type CEGUI::String::rfind (const std::string & std_str, size_type idx = npos) const`  
[inline]

Search backwards for a sub-string.

**Parameters:**

*std\_str* std::string object describing the sub-string to search for

**Note:**

Characters from *std\_str* are considered to represent Unicode code points in the range 0x00..0xFF. No translation of the encountered data is performed.

**Parameters:**

*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *std\_str* travelling backwards from *idx*.
- npos if the sub-string could not be found

**6.282.3.95** `size_type CEGUI::String::find (const utf8 * utf8_str, size_type idx = 0) const`  
[inline]

Search forwards for a sub-string.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the sub-string to search for

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *utf8\_str* travelling forwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.96** `size_type CEGUI::String::rfind (const utf8 * utf8_str, size_type idx = npos) const`  
`[inline]`

Search backwards for a sub-string.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the sub-string to search for

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points `0x00..0x7f`. The use of extended ASCII characters (with values `>0x7f`) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *utf8\_str* travelling backwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.97** `size_type CEGUI::String::find (const utf8 * utf8_str, size_type idx, size_type str_len)`  
`const [inline]`

Search forwards for a sub-string.

**Parameters:**

*utf8\_str* Buffer containing utf8 encoded data describing the sub-string to search for

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points `0x00..0x7f`. The use of extended ASCII characters (with values `>0x7f`) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the code point where the search is to start

*str\_len* Length of the utf8 encoded sub-string in utf8 code units (not code points)

**Returns:**

- Index of the first occurrence of sub-string *utf8\_str* travelling forwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::length\_error* Thrown if *str\_len* is '`npos`'

**6.282.3.98** `size_type CEGUI::String::rfind (const utf8 * utf8_str, size_type idx, size_type str_len) const` `[inline]`

Search backwards for a sub-string.

**Parameters:**

*utf8\_str* Buffer containing utf8 encoded data describing the sub-string to search for

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points `0x00..0x7f`. The use of extended ASCII characters (with values `>0x7f`) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the code point where the search is to start

*str\_len* Length of the utf8 encoded sub-string in utf8 code units (not code points)

**Returns:**

- Index of the first occurrence of sub-string *utf8\_str* travelling backwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::length\_error* Thrown if *str\_len* is '`npos`'

**6.282.3.99** `size_type CEGUI::String::find (const char * cstr, size_type idx = 0) const` `[inline]`

Search forwards for a sub-string.

**Parameters:**

*c\_str* c-string describing the sub-string to search for

*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *c\_str* travelling forwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.100** `size_type CEGUI::String::rfind (const char * cstr, size_type idx = npos) const`  
`[inline]`

Search backwards for a sub-string.

**Parameters:**

*c\_str* c-string describing the sub-string to search for

*idx* Index of the code point where the search is to start

**Returns:**

- Index of the first occurrence of sub-string *c\_str* travelling backwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.101** `size_type CEGUI::String::find (const char * chars, size_type idx, size_type chars_len)`  
`const [inline]`

Search forwards for a sub-string.

**Parameters:**

*chars* char array describing the sub-string to search for

*idx* Index of the code point where the search is to start

*chars\_len* Number of chars in the char array.

**Returns:**

- Index of the first occurrence of sub-string *chars* travelling forwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::length\_error* Thrown if *chars\_len* is '`npos`'

**6.282.3.102** `size_type CEGUI::String::rfind (const char * chars, size_type idx, size_type chars_len) const [inline]`

Search backwards for a sub-string.

**Parameters:**

*chars* char array describing the sub-string to search for

*idx* Index of the code point where the search is to start

*chars\_len* Number of chars in the char array.

**Returns:**

- Index of the first occurrence of sub-string *chars* travelling backwards from *idx*.
- `npos` if the sub-string could not be found

**Exceptions:**

*std::length\_error* Thrown if *chars\_len* is '`npos`'

**6.282.3.103** `size_type CEGUI::String::find_first_of (const String & str, size_type idx = 0) const [inline]`

Find the first occurrence of one of a set of code points.

**Parameters:**

*str* [String](#) object describing the set of code points.

*idx* Index of the start point for the search

**Returns:**

- Index of the first occurrence of any one of the code points in *str* starting from *idx*.
- `npos` if none of the code points in *str* were found.

**6.282.3.104** `size_type CEGUI::String::find_first_not_of (const String & str, size_type idx = 0) const [inline]`

Find the first code point that is not one of a set of code points.

**Parameters:**

*str* [String](#) object describing the set of code points.

*idx* Index of the start point for the search

**Returns:**

- Index of the first code point that does not match any one of the code points in *str* starting from *idx*.
- `npos` if all code points matched one of the code points in *str*.

**6.282.3.105** `size_type CEGUI::String::find_first_of (const std::string & std_str, size_type idx = 0) const` `[inline]`

Find the first occurrence of one of a set of code points.

**Parameters:**

*std\_str* std::string object describing the set of code points.

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the first occurrence of any one of the code points in *std\_str* starting from *idx*.
- npos if none of the code points in *std\_str* were found.

**6.282.3.106** `size_type CEGUI::String::find_first_not_of (const std::string & std_str, size_type idx = 0) const` `[inline]`

Find the first code point that is not one of a set of code points.

**Parameters:**

*std\_str* std::string object describing the set of code points.

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the first code point that does not match any one of the code points in *std\_str* starting from *idx*.
- npos if all code points matched one of the code points in *std\_str*.

**6.282.3.107** `size_type CEGUI::String::find_first_of (const utf8 * utf8_str, size_type idx = 0) const` `[inline]`

Find the first occurrence of one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the set of code points.



**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points `0x00..0x7f`. The use of extended ASCII characters (with values `>0x7f`) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the first occurrence of any one of the code points in *utf8\_str* starting from *idx*.
- `npos` if none of the code points in *utf8\_str* were found.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.108** `size_type CEGUI::String::find_first_not_of (const utf8 * utf8_str, size_type idx = 0)`  
`const` [inline]

Find the first code point that is not one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points `0x00..0x7f`. The use of extended ASCII characters (with values `>0x7f`) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the first code point that does not match any one of the code points in *utf8\_str* starting from *idx*.
- `npos` if all code points matched one of the code points in *utf8\_str*.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.109** `size_type CEGUI::String::find_first_of (const utf8 * utf8_str, size_type idx, size_type str_len) const` [inline]

Find the first occurrence of one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

*str\_len* Length of the utf8 encoded data in utf8 code units (not code points).

**Returns:**

- Index of the first occurrence of any one of the code points in *utf8\_str* starting from *idx*.
- npos if none of the code points in *utf8\_str* were found.

**Exceptions:**

*std::length\_error* Thrown if *str\_len* was 'npos'.

**6.282.3.110** `size_type CEGUI::String::find_first_not_of (const utf8 * utf8_str, size_type idx, size_type str_len) const` `[inline]`

Find the first code point that is not one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

*str\_len* Length of the utf8 encoded data in utf8 code units (not code points).

**Returns:**

- Index of the first code point that does not match any one of the code points in *utf8\_str* starting from *idx*.
- npos if all code points matched one of the code points in *utf8\_str*.

**Exceptions:**

*std::length\_error* Thrown if *str\_len* was 'npos'.

**6.282.3.111** `size_type CEGUI::String::find_first_of (utf32 code_point, size_type idx = 0) const`  
[inline]

Search forwards for a given code point.

**Parameters:**

*code\_point* The utf32 code point to search for  
*idx* Index of the code point where the search is to start.

**Returns:**

- Index of the first occurrence of *code\_point* starting from from *idx*.
- npos if the code point could not be found

**6.282.3.112** `size_type CEGUI::String::find_first_not_of (utf32 code_point, size_type idx = 0) const`  
[inline]

Search forwards for the first code point that does not match a given code point.

**Parameters:**

*code\_point* The utf32 code point to search for  
*idx* Index of the code point where the search is to start.

**Returns:**

- Index of the first code point that does not match *code\_point* starting from from *idx*.
- npos if all code points matched *code\_point*

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.113** `size_type CEGUI::String::find_first_of (const char * cstr, size_type idx = 0) const`  
[inline]

Find the first occurrence of one of a set of chars.

**Parameters:**

*c\_str* c-string describing the set of chars.  
*idx* Index of the start point for the search

**Returns:**

- Index of the first occurrence of any one of the chars in *c\_str* starting from from *idx*.
- npos if none of the chars in *c\_str* were found.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.114** `size_type CEGUI::String::find_first_not_of (const char * cstr, size_type idx = 0) const`  
[inline]

Find the first code point that is not one of a set of chars.

**Parameters:**

*c\_str* c-string describing the set of chars.

*idx* Index of the start point for the search

**Returns:**

- Index of the first code point that does not match any one of the chars in *c\_str* starting from from *idx*.
- npos if all code points matched any of the chars in *c\_str*.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.115** `size_type CEGUI::String::find_first_of (const char * chars, size_type idx, size_type chars_len) const` [inline]

Find the first occurrence of one of a set of chars.

**Parameters:**

*chars* char array containing the set of chars.

*idx* Index of the start point for the search

*chars\_len* Number of chars in the char array.

**Returns:**

- Index of the first occurrence of any one of the chars in *chars* starting from from *idx*.
- npos if none of the chars in *chars* were found.

**Exceptions:**

*std::length\_error* Thrown if *chars\_len* was 'npos'.

**6.282.3.116** `size_type CEGUI::String::find_first_not_of (const char * chars, size_type idx, size_type chars_len) const` [inline]

Find the first code point that is not one of a set of chars.

**Parameters:**

*chars* char array containing the set of chars.

*idx* Index of the start point for the search

*chars\_len* Number of chars in the car array.

**Returns:**

- Index of the first code point that does not match any one of the chars in *chars* starting from *idx*.
- `npos` if all code points matched any of the chars in *chars*.

**Exceptions:**

*std::length\_error* Thrown if *chars\_len* was '`npos`'.

**6.282.3.117** `size_type CEGUI::String::find_last_of (const String & str, size_type idx = npos) const`  
[inline]

Find the last occurrence of one of a set of code points.

**Parameters:**

*str* [String](#) object describing the set of code points.  
*idx* Index of the start point for the search

**Returns:**

- Index of the last occurrence of any one of the code points in *str* starting from *idx*.
- `npos` if none of the code points in *str* were found.

**6.282.3.118** `size_type CEGUI::String::find_last_not_of (const String & str, size_type idx = npos) const`  
[inline]

Find the last code point that is not one of a set of code points.

**Parameters:**

*str* [String](#) object describing the set of code points.  
*idx* Index of the start point for the search

**Returns:**

- Index of the last code point that does not match any one of the code points in *str* starting from *idx*.
- `npos` if all code points matched one of the code points in *str*.

**6.282.3.119** `size_type CEGUI::String::find_last_of (const std::string & std_str, size_type idx = npos) const`  
[inline]

Find the last occurrence of one of a set of code points.

**Parameters:**

*std\_str* `std::string` object describing the set of code points.

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the last occurrence of any one of the code points in *std\_str* starting from *idx*.
- npos if none of the code points in *std\_str* were found.

**6.282.3.120** `size_type CEGUI::String::find_last_not_of (const std::string & std_str, size_type idx = npos) const` [inline]

Find the last code point that is not one of a set of code points.

**Parameters:**

*std\_str* std::string object describing the set of code points.

**Note:**

The characters of *std\_str* are taken to be unencoded data which represent Unicode code points 0x00..0xFF. No translation of the provided data will occur.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the last code point that does not match any one of the code points in *std\_str* starting from *idx*.
- npos if all code points matched one of the code points in *std\_str*.

**6.282.3.121** `size_type CEGUI::String::find_last_of (const utf8 * utf8_str, size_type idx = npos) const` [inline]

Find the last occurrence of one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to utf8\*) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the last occurrence of any one of the code points in *utf8\_str* starting from *idx*.
- *npos* if none of the code points in *utf8\_str* were found.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.122** `size_type CEGUI::String::find_last_not_of (const utf8 * utf8_str, size_type idx = npos) const [inline]`

Find the last code point that is not one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing null-terminated utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

**Returns:**

- Index of the last code point that does not match any one of the code points in *utf8\_str* starting from *idx*.
- *npos* if all code points matched one of the code points in *utf8\_str*.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.123** `size_type CEGUI::String::find_last_of (const utf8 * utf8_str, size_type idx, size_type str_len) const [inline]`

Find the last occurrence of one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to `utf8*`) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

*str\_len* Length of the utf8 encoded data in utf8 code units (not code points).

**Returns:**

- Index of the last occurrence of any one of the code points in *utf8\_str* starting from *idx*.
- *npos* if none of the code points in *utf8\_str* were found.

**Exceptions:**

*std::length\_error* Thrown if *str\_len* was '*npos*'.

**6.282.3.124** `size_type CEGUI::String::find_last_not_of (const utf8 * utf8_str, size_type idx, size_type str_len) const` [inline]

Find the last code point that is not one of a set of code points.

**Parameters:**

*utf8\_str* Buffer containing utf8 encoded data describing the set of code points.

**Note:**

A basic string literal (cast to *utf8\**) can be passed to this function, provided that the string is comprised only of code points 0x00..0x7f. The use of extended ASCII characters (with values >0x7f) would result in incorrect behaviour as the [String](#) will attempt to 'decode' the data, with unpredictable results.

**Parameters:**

*idx* Index of the start point for the search

*str\_len* Length of the utf8 encoded data in utf8 code units (not code points).

**Returns:**

- Index of the last code point that does not match any one of the code points in *utf8\_str* starting from *idx*.
- *npos* if all code points matched one of the code points in *utf8\_str*.

**Exceptions:**

*std::length\_error* Thrown if *str\_len* was '*npos*'.

**6.282.3.125** `size_type CEGUI::String::find_last_of (utf32 code_point, size_type idx = npos) const` [inline]

Search for last occurrence of a given code point.

**Parameters:**

*code\_point* The utf32 code point to search for

*idx* Index of the code point where the search is to start.



**Returns:**

- Index of the last occurrence of *code\_point* starting from *idx*.
- *npos* if the code point could not be found

**6.282.3.126** `size_type CEGUI::String::find_last_not_of (utf32 code_point, size_type idx = npos) const [inline]`

Search for the last code point that does not match a given code point.

**Parameters:**

*code\_point* The utf32 code point to search for  
*idx* Index of the code point where the search is to start.

**Returns:**

- Index of the last code point that does not match *code\_point* starting from *idx*.
- *npos* if all code points matched *code\_point*

**6.282.3.127** `size_type CEGUI::String::find_last_of (const char * cstr, size_type idx = npos) const [inline]`

Find the last occurrence of one of a set of chars.

**Parameters:**

*c\_str* c-string describing the set of chars.  
*idx* Index of the start point for the search

**Returns:**

- Index of the last occurrence of any one of the chars in *c\_str* starting from *idx*.
- *npos* if none of the chars in *c\_str* were found.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.128** `size_type CEGUI::String::find_last_not_of (const char * cstr, size_type idx = npos) const [inline]`

Find the last code point that is not one of a set of chars.

**Parameters:**

*c\_str* c-string describing the set of chars.  
*idx* Index of the start point for the search

**Returns:**

- Index of the last code point that does not match any one of the chars in *c\_str* starting from *idx*.

- `npos` if all code points matched any of the chars in *c\_str*.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.129** `size_type CEGUI::String::find_last_of (const char * chars, size_type idx, size_type chars_len) const` `[inline]`

Find the last occurrence of one of a set of chars.

**Parameters:**

*chars* char array containing the set of chars.  
*idx* Index of the start point for the search  
*chars\_len* Number of chars in the char array.

**Returns:**

- Index of the last occurrence of any one of the chars in *chars*, starting from *idx*.
- `npos` if none of the chars in *chars* were found.

**Exceptions:**

*std::length\_error* Thrown if *chars\_len* was '`npos`'.

**6.282.3.130** `size_type CEGUI::String::find_last_not_of (const char * chars, size_type idx, size_type chars_len) const` `[inline]`

Find the last code point that is not one of a set of chars.

**Parameters:**

*chars* char array containing the set of chars.  
*idx* Index of the start point for the search  
*chars\_len* Number of chars in the char array.

**Returns:**

- Index of the last code point that does not match any one of the chars in *chars*, starting from *idx*.
- `npos` if all code points matched any of the chars in *chars*.

**Exceptions:**

*std::length\_error* Thrown if *chars\_len* was '`npos`'.

**6.282.3.131** `String CEGUI::String::substr (size_type idx = 0, size_type len = npos) const` `[inline]`

Returns a substring of this [String](#).

**Parameters:**

*idx* Index of the first code point to use for the sub-string.

*len* Maximum number of code points to use for the sub-string

**Returns:**

A [String](#) object containing the specified sub-string.

**Exceptions:**

*std::out\_of\_range* Thrown if *idx* is invalid for this [String](#).

**6.282.3.132** `iterator CEGUI::String::begin (void)` `[inline]`

Return a forwards [iterator](#) that describes the beginning of the [String](#).

**Returns:**

[iterator](#) object that describes the beginning of the [String](#).

**6.282.3.133** `const_iterator CEGUI::String::begin (void) const` `[inline]`

Return a constant forwards [iterator](#) that describes the beginning of the [String](#).

**Returns:**

[const\\_iterator](#) object that describes the beginning of the [String](#).

**6.282.3.134** `iterator CEGUI::String::end (void)` `[inline]`

Return a forwards [iterator](#) that describes the end of the [String](#).

**Returns:**

[iterator](#) object that describes the end of the [String](#).

**6.282.3.135** `const_iterator CEGUI::String::end (void) const` `[inline]`

Return a constant forwards [iterator](#) that describes the end of the [String](#).

**Returns:**

[const\\_iterator](#) object that describes the end of the [String](#).

**6.282.3.136 reverse\_iterator CEGUI::String::rbegin (void) [inline]**

Return a reverse [iterator](#) that describes the beginning of the [String](#).

**Returns:**

reverse\_iterator object that describes the beginning of the [String](#) (so is actually at the end)

**6.282.3.137 const\_reverse\_iterator CEGUI::String::rbegin (void) const [inline]**

Return a constant reverse [iterator](#) that describes the beginning of the [String](#).

**Returns:**

const\_reverse\_iterator object that describes the beginning of the [String](#) (so is actually at the end)

**6.282.3.138 reverse\_iterator CEGUI::String::rend (void) [inline]**

Return a reverse [iterator](#) that describes the end of the [String](#).

**Returns:**

reverse\_iterator object that describes the end of the [String](#) (so is actually at the beginning)

**6.282.3.139 const\_reverse\_iterator CEGUI::String::rend (void) const [inline]**

Return a constant reverse [iterator](#) that describes the end of the [String](#).

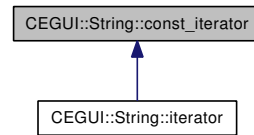
**Returns:**

const\_reverse\_iterator object that describes the end of the [String](#) (so is actually at the beginning)

## 6.283 CEGUI::String::const\_iterator Class Reference

Constant forward [iterator](#) class for [String](#) objects.

Inheritance diagram for CEGUI::String::const\_iterator:



### Public Member Functions

- **const\_iterator** ([const\\_pointer](#) ptr)
- **const\_reference operator \*** () const
- **const\_pointer operator →** () const
- **const\_iterator & operator++** ()
- **const\_iterator operator++** (int)
- **const\_iterator & operator--** ()
- **const\_iterator operator--** (int)
- **const\_iterator & operator+=** ([difference\\_type](#) offset)
- **const\_iterator operator+** ([difference\\_type](#) offset) const
- **const\_iterator & operator-=** ([difference\\_type](#) offset)
- **const\_iterator operator-** ([difference\\_type](#) offset) const
- **difference\_type operator-** (const [const\\_iterator](#) &iter) const
- **const\_reference operator[]** ([difference\\_type](#) offset) const
- **bool operator==** (const [const\\_iterator](#) &iter) const
- **bool operator!=** (const [const\\_iterator](#) &iter) const
- **bool operator<** (const [const\\_iterator](#) &iter) const
- **bool operator>** (const [const\\_iterator](#) &iter) const
- **bool operator<=** (const [const\\_iterator](#) &iter) const
- **bool operator>=** (const [const\\_iterator](#) &iter) const

### Public Attributes

- **const utf32 \* d\_ptr**

### Friends

- **const\_iterator operator+** ([difference\\_type](#) offset, const [const\\_iterator](#) &iter)

#### 6.283.1 Detailed Description

Constant forward [iterator](#) class for [String](#) objects.

## 6.284 CEGUI::String::FastLessCompare Struct Reference

Functor that can be used as comparator in a `std::map` with [String](#) keys. It's faster than using the default, but the map will no longer be sorted alphabetically.

### Public Member Functions

- `bool operator() (const String &a, const String &b) const`

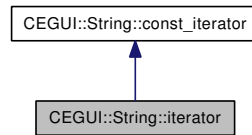
#### 6.284.1 Detailed Description

Functor that can be used as comparator in a `std::map` with [String](#) keys. It's faster than using the default, but the map will no longer be sorted alphabetically.

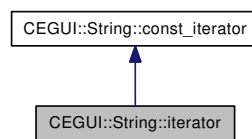
## 6.285 CEGUI::String::iterator Class Reference

Forward [iterator](#) class for [String](#) objects.

Inheritance diagram for CEGUI::String::iterator:



Collaboration diagram for CEGUI::String::iterator:



### Public Member Functions

- [iterator](#) ([pointer](#) ptr)
- [reference operator](#) \* () const
- [pointer operator](#) → () const
- [iterator](#) & [operator++](#) ()
- [iterator](#) [operator++](#) (int)
- [iterator](#) & [operator--](#) ()
- [iterator](#) [operator--](#) (int)
- [iterator](#) & [operator+=](#) ([difference\\_type](#) offset)
- [iterator](#) [operator+](#) ([difference\\_type](#) offset) const
- [iterator](#) & [operator-=](#) ([difference\\_type](#) offset)
- [iterator](#) [operator-](#) ([difference\\_type](#) offset) const
- [difference\\_type](#) [operator-](#) (const [const\\_iterator](#) &iter) const
- [reference operator](#) [ ] ([difference\\_type](#) offset) const

### Friends

- [iterator](#) [operator+](#) ([difference\\_type](#) offset, const [iterator](#) &iter)

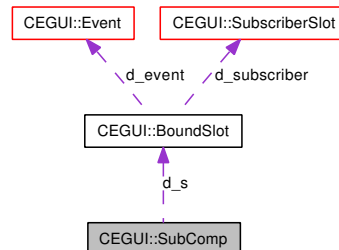
#### 6.285.1 Detailed Description

Forward [iterator](#) class for [String](#) objects.

## 6.286 CEGUI::SubComp Class Reference

Implementation helper functor which is used to locate a [BoundSlot](#) in the multimap collection of Bound-Slots.

Collaboration diagram for CEGUI::SubComp:



### Public Member Functions

- **SubComp** (const [BoundSlot](#) &s)
- **operator()** (std::pair< [Event::Group](#), [Event::Connection](#) > e) const

### 6.286.1 Detailed Description

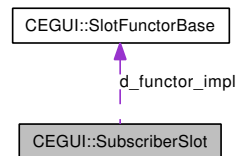
Implementation helper functor which is used to locate a [BoundSlot](#) in the multimap collection of Bound-Slots.



## 6.287 CEGUI::SubscriberSlot Class Reference

[SubscriberSlot](#) class which is used when subscribing to events.

Collaboration diagram for CEGUI::SubscriberSlot:



### Public Member Functions

- [SubscriberSlot](#) ()  
*Default constructor. Creates a [SubscriberSlot](#) with no bound slot.*
- [SubscriberSlot](#) (FreeFunctionSlot::SlotFunction \*func)  
*Creates a [SubscriberSlot](#) that is bound to a free function.*
- [~SubscriberSlot](#) ()  
*Destructor. Note this is non-virtual, which should be telling you not to sub-class!*
- bool [operator\(\)](#) (const [EventArgs](#) &args) const  
*Invokes the slot functor that is bound to this Subscriber. Returns whatever the slot returns, unless there is not slot bound when false is always returned.*
- bool [connected](#) () const  
*Returns whether the [SubscriberSlot](#) is internally connected (bound).*
- void [cleanup](#) ()  
*Disconnects the slot internally and performs any required cleanup operations.*
- template<typename T>  
[SubscriberSlot](#) (bool(T::\*function)(const [EventArgs](#) &), T \*obj)  
*Creates a [SubscriberSlot](#) that is bound to a member function.*
- template<typename T>  
[SubscriberSlot](#) (const [FunctorReferenceBinder](#)< T > &binder)  
*Creates a [SubscriberSlot](#) that is bound to a functor object reference.*
- template<typename T>  
[SubscriberSlot](#) (const T &functor)  
*Creates a [SubscriberSlot](#) that is bound to a copy of a functor object.*
- template<typename T>  
[SubscriberSlot](#) (T \*functor)  
*Creates a [SubscriberSlot](#) that is bound to a functor pointer.*

### 6.287.1 Detailed Description

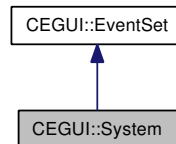
[SubscriberSlot](#) class which is used when subscribing to events.

For many uses, the construction of the [SubscriberSlot](#) may be implicit, so you do not have to specify Subscriber in your subscription calls. Notable exceptions are for subscribing member functions and references to functor objects.

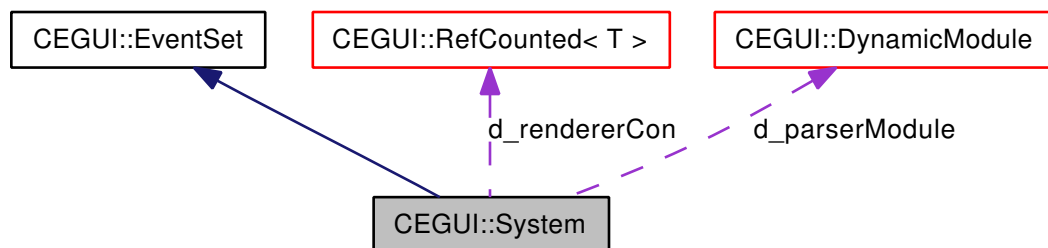
## 6.288 CEGUI::System Class Reference

The [System](#) class is the [CEGUI](#) class that provides access to all other elements in this system.

Inheritance diagram for CEGUI::System:



Collaboration diagram for CEGUI::System:



### Public Member Functions

- [System](#) ([Renderer](#) \*renderer, [ResourceProvider](#) \*resourceProvider=0, [XMLParser](#) \*xmlParser=0, [ScriptModule](#) \*scriptModule=0, const [String](#) &configFile="", const [String](#) &log-File="CEGUI.log")  
Construct a new [System](#) object.
- [~System](#) (void)  
Destructor for [System](#) objects.
- [Renderer](#) \* [getRenderer](#) (void) const  
Return a pointer to the [Renderer](#) object being used by the system.
- void [setDefaultFont](#) (const [String](#) &name)  
Set the default font to be used by the system.
- void [setDefaultFont](#) ([Font](#) \*font)  
Set the default font to be used by the system.
- [Font](#) \* [getDefaultFont](#) (void) const  
Return a pointer to the default [Font](#) for the GUI system.
- void [signalRedraw](#) ()  
Causes a full re-draw next time [renderGUI\(\)](#) is called.
- bool [isRedrawRequested](#) () const

Return a boolean value to indicate whether a full re-draw is requested next time [renderGUI\(\)](#) is called.

- void [renderGUI](#) (void)  
*Render the GUI.*
- [Window](#) \* [setGUISheet](#) ([Window](#) \*sheet)  
*Set the active GUI sheet (root) window.*
- [Window](#) \* [getGUISheet](#) (void) const  
*Return a pointer to the active GUI sheet (root) window.*
- double [getSingleClickTimeout](#) (void) const  
*Return the current timeout for generation of single-click events.*
- double [getMultiClickTimeout](#) (void) const  
*Return the current timeout for generation of multi-click events.*
- const [Size](#) & [getMultiClickToleranceAreaSize](#) (void) const  
*Return the size of the allowable mouse movement tolerance used when generating multi-click events.*
- void [setSingleClickTimeout](#) (double timeout)  
*Set the timeout used for generation of single-click events.*
- void [setMultiClickTimeout](#) (double timeout)  
*Set the timeout to be used for the generation of multi-click events.*
- void [setMultiClickToleranceAreaSize](#) (const [Size](#) &sz)  
*Set the size of the allowable mouse movement tolerance used when generating multi-click events.*
- const [Image](#) \* [getDefaultMouseCursor](#) (void) const  
*Return the currently set default mouse cursor image.*
- void [setDefaultMouseCursor](#) (const [Image](#) \*image)  
*Set the image to be used as the default mouse cursor.*
- void [setDefaultMouseCursor](#) ([MouseCursorImage](#) image)  
*Set the image to be used as the default mouse cursor.*
- void [setDefaultMouseCursor](#) (const [String](#) &imageset, const [String](#) &image\_name)  
*Set the image to be used as the default mouse cursor.*
- [Window](#) \* [getWindowContainingMouse](#) (void) const  
*Return the [Window](#) object that the mouse is presently within.*
- [ScriptModule](#) \* [getScriptingModule](#) (void) const  
*Return a pointer to the [ScriptModule](#) being used for scripting within the GUI system.*
- void [setScriptingModule](#) ([ScriptModule](#) \*scriptModule)  
*Set the [ScriptModule](#) to be used for scripting within the GUI system.*

- **ResourceProvider \* getResourceProvider** (void) const  
*Return a pointer to the **ResourceProvider** being used within the GUI system.*
- void **executeScriptFile** (const **String** &filename, const **String** &resourceGroup="") const  
*Execute a script file if possible.*
- int **executeScriptGlobal** (const **String** &function\_name) const  
*Execute a scripted global function if possible. The function should not take any parameters and should return an integer.*
- void **executeScriptString** (const **String** &str) const  
*If possible, execute script code contained in the given **CEGUI::String** object.*
- float **getMouseMoveScaling** (void) const  
*return the current mouse movement scaling factor.*
- void **setMouseMoveScaling** (float scaling)  
*Set the current mouse movement scaling factor.*
- void **notifyWindowDestroyed** (const **Window** \*window)  
*Internal method used to inform the **System** object whenever a window is destroyed, so that **System** can perform any required housekeeping.*
- uint **getSystemKeys** (void) const  
*Return the current system keys value.*
- **XMLParser \* getXMLParser** (void) const  
*Return the **XMLParser** object.*
- void **setDefaultTooltip** (**Tooltip** \*tooltip)  
*Set the system default **Tooltip** object. This value may be NULL to indicate that no default **Tooltip** will be available.*
- void **setDefaultTooltip** (const **String** &tooltipType)  
*Set the system default **Tooltip** to be used by specifying a **Window** type.*
- **Tooltip \* getDefaultTooltip** (void) const  
*return a pointer to the system default tooltip. May return 0.*
- void **setModalTarget** (**Window** \*target)  
*Internal method to directly set the current modal target.*
- **Window \* getModalTarget** (void) const  
*Return a pointer to the **Window** that is currently the modal target.*
- bool **injectMouseMove** (float delta\_x, float delta\_y)  
*Method that injects a mouse movement event into the system.*
- bool **injectMouseLeaves** (void)  
*Method that injects that the mouse has left the application window.*

- bool [injectMouseButtonDown](#) ([MouseButton](#) button)  
*Method that injects a mouse button down event into the system.*
- bool [injectMouseButtonUp](#) ([MouseButton](#) button)  
*Method that injects a mouse button up event into the system.*
- bool [injectKeyDown](#) (uint key\_code)  
*Method that injects a key down event into the system.*
- bool [injectKeyUp](#) (uint key\_code)  
*Method that injects a key up event into the system.*
- bool [injectChar](#) (utf32 code\_point)  
*Method that injects a typed character event into the system.*
- bool [injectMouseWheelChange](#) (float delta)  
*Method that injects a mouse-wheel / scroll-wheel event into the system.*
- bool [injectMousePosition](#) (float x\_pos, float y\_pos)  
*Method that injects a new position for the mouse cursor.*
- bool [injectTimePulse](#) (float timeElapsed)  
*Method to inject time pulses into the system.*

## Static Public Member Functions

- static [System](#) & [getSingleton](#) (void)  
*Return singleton [System](#) object.*
- static [System](#) \* [getSingletonPtr](#) (void)  
*Return pointer to singleton [System](#) object.*
- static void [setDefaultXMLParserName](#) (const [String](#) &parserName)  
*Static member to set the name of the default XML parser module that should be used.*
- static const [String](#) [getDefaultXMLParserName](#) ()  
*Return the name of the currently set default xml parser module.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const double [DefaultSingleClickTimeout](#) = 0.2  
*Default timeout for generation of single click events.*

- static const double [DefaultMultiClickTimeout](#) = 0.33  
*Default timeout for generation of multi-click events.*
- static const [Size](#) [DefaultMultiClickAreaSize](#)  
*Default allowable mouse movement for multi-click event generation.*
- static const [String](#) [EventGUISheetChanged](#)  
*Name of event fired whenever the GUI sheet is changed.*
- static const [String](#) [EventSingleClickTimeoutChanged](#)  
*Name of event fired when the single-click timeout is changed.*
- static const [String](#) [EventMultiClickTimeoutChanged](#)  
*Name of event fired when the multi-click timeout is changed.*
- static const [String](#) [EventMultiClickAreaSizeChanged](#)  
*Name of event fired when the size of the multi-click tolerance area is changed.*
- static const [String](#) [EventDefaultFontChanged](#)  
*Name of event fired when the default font changes.*
- static const [String](#) [EventDefaultMouseCursorChanged](#)  
*Name of event fired when the default mouse cursor changes.*
- static const [String](#) [EventMouseMoveScalingChanged](#)  
*Name of event fired when the mouse move scaling factor changes.*

### 6.288.1 Detailed Description

The [System](#) class is the [CEGUI](#) class that provides access to all other elements in this system.

This object must be created by the client application. The [System](#) object requires that you pass it an initialised [Renderer](#) object which it can use to interface to whatever rendering system will be used to display the GUI imagery.

### 6.288.2 Constructor & Destructor Documentation

**6.288.2.1 CEGUI::System::System (Renderer \* *renderer*, ResourceProvider \* *resourceProvider* = 0, XMLParser \* *xmlParser* = 0, ScriptModule \* *scriptModule* = 0, const String & *configFile* = "", const String & *logFile* = "CEGUI.log")**

Construct a new [System](#) object.

#### Parameters:

***renderer*** Pointer to the valid [Renderer](#) object that will be used to render GUI imagery.

***resourceProvider*** Pointer to a [ResourceProvider](#) object, or NULL to use whichever default the [Renderer](#) provides.

***xmlParser*** Pointer to a valid [XMLParser](#) object to be used when parsing XML files, or NULL to use a default parser.

*scriptModule* Pointer to a [ScriptModule](#) object. may be NULL for none.

*configFile* [String](#) object containing the name of a configuration file to use.

*logFile* [String](#) object containing the name to use for the log file.

### 6.288.3 Member Function Documentation

#### 6.288.3.1 `Renderer* CEGUI::System::getRenderer (void) const` [inline]

Return a pointer to the [Renderer](#) object being used by the system.

**Returns:**

Pointer to the [Renderer](#) object used by the system.

#### 6.288.3.2 `System & CEGUI::System::getSingleton (void)` [static]

Return singleton [System](#) object.

**Returns:**

Singleton [System](#) object

#### 6.288.3.3 `System * CEGUI::System::getSingletonPtr (void)` [static]

Return pointer to singleton [System](#) object.

**Returns:**

Pointer to singleton [System](#) object

#### 6.288.3.4 `void CEGUI::System::setDefaultFont (const String & name)`

Set the default font to be used by the system.

**Parameters:**

*name* [String](#) object containing the name of the font to be used as the system default.

**Returns:**

Nothing.

#### 6.288.3.5 `void CEGUI::System::setDefaultFont (Font * font)`

Set the default font to be used by the system.

**Parameters:**

*font* Pointer to the font to be used as the system default.

**Returns:**

Nothing.



**6.288.3.6** `Font* CEGUI::System::getDefaultFont (void) const` `[inline]`

Return a pointer to the default [Font](#) for the GUI system.

**Returns:**

Pointer to a [Font](#) object that is the default font in the system.

**6.288.3.7** `void CEGUI::System::signalRedraw ()` `[inline]`

Causes a full re-draw next time [renderGUI\(\)](#) is called.

**Returns:**

Nothing

**6.288.3.8** `bool CEGUI::System::isRedrawRequested () const` `[inline]`

Return a boolean value to indicate whether a full re-draw is requested next time [renderGUI\(\)](#) is called.

**Returns:**

true if a re-draw has been requested

**6.288.3.9** `void CEGUI::System::renderGUI (void)`

Render the GUI.

Depending upon the internal state, this may either re-use rendering from last time, or trigger a full re-draw from all elements.

**Returns:**

Nothing

**6.288.3.10** `Window * CEGUI::System::setGUISheet (Window * sheet)`

Set the active GUI sheet (root) window.

**Parameters:**

*sheet* Pointer to a [Window](#) object that will become the new GUI 'root'

**Returns:**

Pointer to the window that was previously set as the GUI root.

**6.288.3.11 Window\* CEGUI::System::getGUISheet (void) const** `[inline]`

Return a pointer to the active GUI sheet (root) window.

**Returns:**

Pointer to the window object that has been set as the GUI root element.

**6.288.3.12 double CEGUI::System::getSingleClickTimeout (void) const** `[inline]`

Return the current timeout for generation of single-click events.

A single-click is defined here as a button being pressed and then released.

**Returns:**

double value equal to the current single-click timeout value.

**6.288.3.13 double CEGUI::System::getMultiClickTimeout (void) const** `[inline]`

Return the current timeout for generation of multi-click events.

A multi-click event is a double-click, or a triple-click. The value returned here is the maximum allowable time between mouse button down events for which a multi-click event will be generated.

**Returns:**

double value equal to the current multi-click timeout value.

**6.288.3.14 const Size& CEGUI::System::getMultiClickToleranceAreaSize (void) const**  
`[inline]`

Return the size of the allowable mouse movement tolerance used when generating multi-click events.

This size defines an area with the mouse at the centre. The mouse must stay within the tolerance defined for a multi-click (double click, or triple click) event to be generated.

**Returns:**

[Size](#) object describing the current multi-click tolerance area size.

**6.288.3.15 void CEGUI::System::setSingleClickTimeout (double *timeout*)**

Set the timeout used for generation of single-click events.

A single-click is defined here as a button being pressed and then released.

**Parameters:**

*timeout* double value equal to the single-click timeout value to be used from now onwards.

**Returns:**

Nothing.

**6.288.3.16 void CEGUI::System::setMultiClickTimeout (double *timeout*)**

Set the timeout to be used for the generation of multi-click events.

A multi-click event is a double-click, or a triple-click. The value returned here is the maximum allowable time between mouse button down events for which a multi-click event will be generated.

**Parameters:**

*timeout* double value equal to the multi-click timeout value to be used from now onwards.

**Returns:**

Nothing.

**6.288.3.17 void CEGUI::System::setMultiClickToleranceAreaSize (const Size & *sz*)**

Set the size of the allowable mouse movement tolerance used when generating multi-click events.

This size defines an area with the mouse at the centre. The mouse must stay within the tolerance defined for a multi-click (double click, or triple click) event to be generated.

**Parameters:**

*sz* [Size](#) object describing the multi-click tolerance area size to be used.

**Returns:**

Nothing.

**6.288.3.18 const Image\* CEGUI::System::getDefaultMouseCursor (void) const [inline]**

Return the currently set default mouse cursor image.

**Returns:**

Pointer to the current default image used for the mouse cursor. May return NULL if default cursor has not been set, or has intentionally been set to NULL - which results in a blank default cursor.

**6.288.3.19 void CEGUI::System::setDefaultMouseCursor (const Image \* *image*)**

Set the image to be used as the default mouse cursor.

**Parameters:**

*image* Pointer to an image object that is to be used as the default mouse cursor. To have no cursor rendered by default, you can specify NULL here.

**Returns:**

Nothing.

**6.288.3.20** `void CEGUI::System::setDefaultMouseCursor (MouseCursorImage image)`  
[inline]

Set the image to be used as the default mouse cursor.

**Parameters:**

*image* One of the MouseCursorImage enumerated values.

**Returns:**

Nothing.

**6.288.3.21** `void CEGUI::System::setDefaultMouseCursor (const String & imageset, const String & image_name)`

Set the image to be used as the default mouse cursor.

**Parameters:**

*imageset* [String](#) object that contains the name of the [Imageset](#) that contains the image to be used.

*image\_name* [String](#) object that contains the name of the [Image](#) on *imageset* that is to be used.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if *imageset* is not known, or if *imageset* contains no [Image](#) named *image\_name*.

**6.288.3.22** `Window* CEGUI::System::getWindowContainingMouse (void) const` [inline]

Return the [Window](#) object that the mouse is presently within.

**Returns:**

Pointer to the [Window](#) object that currently contains the mouse cursor, or NULL if none.

**6.288.3.23** `ScriptModule * CEGUI::System::getScriptingModule (void) const`

Return a pointer to the [ScriptModule](#) being used for scripting within the GUI system.

**Returns:**

Pointer to a [ScriptModule](#) based object.

**6.288.3.24 void CEGUI::System::setScriptingModule (ScriptModule \* *scriptModule*)**

Set the [ScriptModule](#) to be used for scripting within the GUI system.

**Parameters:**

*scriptModule* Pointer to a [ScriptModule](#) based object, or 0 for none (be careful!)

**Returns:**

Nothing

**6.288.3.25 ResourceProvider \* CEGUI::System::getResourceProvider (void) const**

Return a pointer to the [ResourceProvider](#) being used within the GUI system.

**Returns:**

Pointer to a [ResourceProvider](#) based object.

**6.288.3.26 void CEGUI::System::executeScriptFile (const String & *filename*, const String & *resourceGroup* = " ") const**

Execute a script file if possible.

**Parameters:**

*filename* [String](#) object holding the filename of the script file that is to be executed

*resourceGroup* Resource group identifier to be passed to the [ResourceProvider](#) when loading the script file.

**6.288.3.27 int CEGUI::System::executeScriptGlobal (const String & *function\_name*) const**

Execute a scripted global function if possible. The function should not take any parameters and should return an integer.

**Parameters:**

*function\_name* [String](#) object holding the name of the function, in the global script environment, that is to be executed.

**Returns:**

The integer value returned from the script function.

**6.288.3.28 void CEGUI::System::executeScriptString (const String & *str*) const**

If possible, execute script code contained in the given [CEGUI::String](#) object.

**Parameters:**

*str* [String](#) object holding the valid script code that should be executed.

**Returns:**

Nothing.

**6.288.3.29 float CEGUI::System::getMouseMoveScaling (void) const**

return the current mouse movement scaling factor.

**Returns:**

float value that is equal to the currently set mouse movement scaling factor. Defaults to 1.0f.

**6.288.3.30 void CEGUI::System::setMouseMoveScaling (float *scaling*)**

Set the current mouse movement scaling factor.

**Parameters:**

*scaling* float value specifying the scaling to be applied to mouse movement inputs.

**Returns:**

nothing.

**6.288.3.31 void CEGUI::System::notifyWindowDestroyed (const Window \* *window*)**

Internal method used to inform the [System](#) object whenever a window is destroyed, so that [System](#) can perform any required housekeeping.

**Note:**

This method is not intended for client code usage. If you use this method anything can, and probably will, go wrong!

**6.288.3.32 uint CEGUI::System::getSystemKeys (void) const [inline]**

Return the current system keys value.

**Returns:**

uint value representing a combination of the SystemKey bits.

**6.288.3.33 void CEGUI::System::setDefaultTooltip (Tooltip \* *tooltip*)**

Set the system default [Tooltip](#) object. This value may be NULL to indicate that no default [Tooltip](#) will be available.

**Parameters:**

*tooltip* Pointer to a valid [Tooltip](#) based object which should be used as the default tooltip for the system, or NULL to indicate that no system default tooltip is required. Note that when passing a pointer to a [Tooltip](#) object, ownership of the [Tooltip](#) does not pass to [System](#).

**Returns:**

Nothing.

**6.288.3.34 void CEGUI::System::setDefaultTooltip (const String & *tooltipType*)**

Set the system default [Tooltip](#) to be used by specifying a [Window](#) type.

[System](#) will internally attempt to create an instance of the specified window type (which must be derived from the base [Tooltip](#) class). If the [Tooltip](#) creation fails, the error is logged and no system default [Tooltip](#) will be available.

**Parameters:**

*tooltipType* [String](#) object holding the name of the [Tooltip](#) based [Window](#) type which should be used as the [Tooltip](#) for the system default.

**Returns:**

Nothing.

**6.288.3.35 Tooltip\* CEGUI::System::getDefaultTooltip (void) const [inline]**

return a pointer to the system default tooltip. May return 0.

**Returns:**

Pointer to the current system default tooltip. May return 0 if no system default tooltip is available.

**6.288.3.36 void CEGUI::System::setModalTarget (Window \* *target*) [inline]**

Internal method to directly set the current modal target.

**Note:**

This method is called internally by [Window](#), and must be used by client code. Doing so will most likely not have the expected results.

**6.288.3.37 Window\* CEGUI::System::getModalTarget (void) const** [inline]

Return a pointer to the [Window](#) that is currently the modal target.

**Returns:**

Pointer to the current modal target. NULL if there is no modal target.

**6.288.3.38 void CEGUI::System::setDefaultXMLParserName (const String & parserName)**  
[static]

Static member to set the name of the default XML parser module that should be used.

If you want to modify the default parser from the one compiled in, you need to call this static member prior to instantiating the main [CEGUI::System](#) object.

Note that calling this member to change the name of the default module after [CEGUI::System](#), and therefore the default xml parser, has been created will have no real effect - the default parser name will be updated, though no actual changes to the xml parser module will occur.

The built-in options for this are:

- XercesParser
- ExpatParser
- LibxmlParser
- TinyXMLParser

Whether these are actually available, depends upon how you built the system. If you have some custom parser, you can provide the name of that here to have it used as the default, though note that the final filename of the parser module should be of the form:

[prefix][CEGUI](#)[parserName][suffix]

where:

- [prefix] is some optional prefix; like 'lib' on linux.
- [CEGUI](#) is a required prefix.
- [parserName] is the name of the parser, as supplied to this function.
- [suffix] is the filename suffix, like .dll or .so

Final module filenames are, thus, of the form:

- CEGUIXercesParser.dll
- libCEGUIXercesParser.so

**Parameters:**

*parserName* [String](#) describing the name of the xml parser module to be used as the default.

**Returns:**

Nothing.



**6.288.3.39** `const String CEGUI::System::getDefaultXMLParserName ()` `[static]`

Return the name of the currently set default xml parser module.

**Returns:**

[String](#) holding the currently set default xml parser name. Note that if this name has been changed after instantiating the system, the name returned may not actually correspond to the module in use.

**6.288.3.40** `bool CEGUI::System::injectMouseMove (float delta_x, float delta_y)`

Method that injects a mouse movement event into the system.

**Parameters:**

*delta\_x* amount the mouse moved on the x axis.

*delta\_y* amount the mouse moved on the y axis.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.41** `bool CEGUI::System::injectMouseLeaves (void)`

Method that injects that the mouse has left the application window.

**Returns:**

- true if the generated mouse move event was handled.
- false if the generated mouse move event was not handled.

**6.288.3.42** `bool CEGUI::System::injectMouseButtonDown (MouseButton button)`

Method that injects a mouse button down event into the system.

**Parameters:**

*button* One of the MouseButton values indicating which button was pressed.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.43 bool CEGUI::System::injectMouseButtonUp (MouseButton *button*)**

Method that injects a mouse button up event into the system.

**Parameters:**

*button* One of the MouseButton values indicating which button was released.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.44 bool CEGUI::System::injectKeyDown (uint *key\_code*)**

Method that injects a key down event into the system.

**Parameters:**

*key\_code* uint value indicating which key was pressed.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.45 bool CEGUI::System::injectKeyUp (uint *key\_code*)**

Method that injects a key up event into the system.

**Parameters:**

*key\_code* uint value indicating which key was released.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.46 bool CEGUI::System::injectChar (utf32 *code\_point*)**

Method that injects a typed character event into the system.

**Parameters:**

*code\_point* Unicode code point of the character that was typed.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.47 bool CEGUI::System::injectMouseWheelChange (float *delta*)**

Method that injects a mouse-wheel / scroll-wheel event into the system.

**Parameters:**

*delta* float value representing the amount the wheel moved.

**Returns:**

- true if the input was processed by the gui system.
- false if the input was not processed by the gui system.

**6.288.3.48 bool CEGUI::System::injectMousePosition (float *x\_pos*, float *y\_pos*)**

Method that injects a new position for the mouse cursor.

**Parameters:**

*x\_pos* New absolute pixel position of the mouse cursor on the x axis.

*y\_pos* New absolute pixel position of the mouse cursor in the y axis.

**Returns:**

- true if the generated mouse move event was handled.
- false if the generated mouse move event was not handled.

**6.288.3.49 bool CEGUI::System::injectTimePulse (float *timeElapsed*)**

Method to inject time pulses into the system.

**Parameters:**

*timeElapsed* float value indicating the amount of time passed, in seconds, since the last time this method was called.

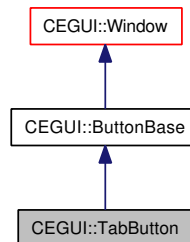
**Returns:**

Currently, this method always returns true.

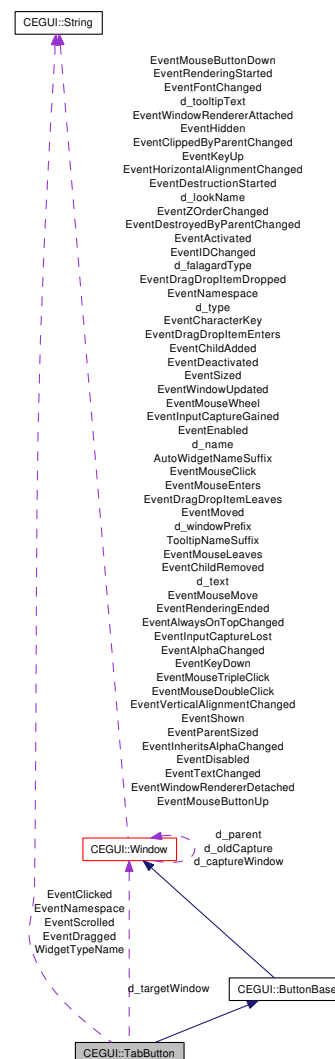
## 6.289 CEGUI::TabButton Class Reference

Base class for TabButtons. A [TabButton](#) based class is used internally as the button that appears at the top of a [TabControl](#) widget to select the active tab pane.

Inheritance diagram for CEGUI::TabButton:



Collaboration diagram for CEGUI::TabButton:



## Public Member Functions

- **TabButton** (const **String** &type, const **String** &name)  
*Constructor for base **TabButton** class.*
- virtual **~TabButton** (void)  
*Destructor for **TabButton** class.*
- virtual void **setSelected** (bool selected)  
*Set whether this tab button is selected or not.*
- bool **isSelected** (void) const  
*Return whether this tab button is selected or not.*
- void **setTargetWindow** (**Window** \*wnd)  
*Set the target window which is the content pane which this button is covering.*
- **Window** \* **getTargetWindow** (void)  
*Get the target window which is the content pane which this button is covering.*

## Static Public Attributes

- static const **String** **EventNamespace**  
*Namespace for global events.*
- static const **String** **WidgetTypeName**  
***Window** factory name.*
- static const **String** **EventClicked**  
*The button was clicked.*
- static const **String** **EventDragged**  
*Attempt to drag the button with middle button.*
- static const **String** **EventScrolled**  
*Scroll wheel activated on top of the button.*

## Protected Member Functions

- virtual void **onClicked** (**WindowEventArgs** &e)  
*handler invoked internally when the button is clicked.*
- virtual void **onMouseButtonUp** (**MouseEventArgs** &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void **onMouseDown** (**MouseEventArgs** &e)  
*Handler called when a mouse button has been depressed within this window's area.*

- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*
- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

## Protected Attributes

- bool [d\\_selected](#)  
*Is this button selected?*
- bool [d\\_dragging](#)  
*In drag mode or not.*
- [Window](#) \* [d\\_targetWindow](#)  
*The target window which this button is representing.*

### 6.289.1 Detailed Description

Base class for TabButtons. A [TabButton](#) based class is used internally as the button that appears at the top of a [TabControl](#) widget to select the active tab pane.

### 6.289.2 Member Function Documentation

#### 6.289.2.1 void CEGUI::TabButton::onMouseButtonUp ([MouseEventArgs](#) & *e*) [protected, virtual]

Handler called when a mouse button has been released within this window's area.

##### Parameters:

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

#### 6.289.2.2 void CEGUI::TabButton::onMouseButtonDown ([MouseEventArgs](#) & *e*) [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

##### Parameters:

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

**6.289.2.3** `void CEGUI::TabButton::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.289.2.4** `void CEGUI::TabButton::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

**6.289.2.5** `virtual bool CEGUI::TabButton::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

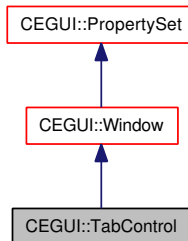
true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::ButtonBase](#).

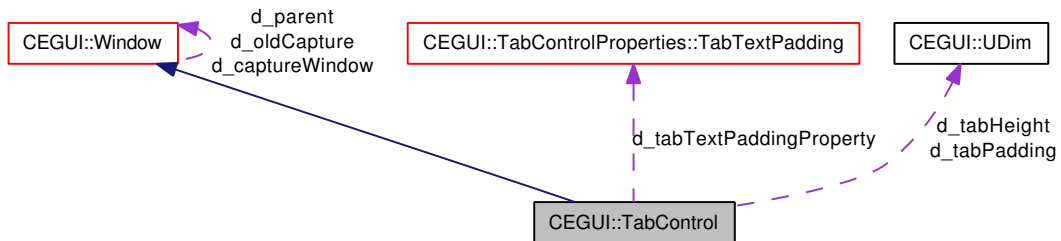
## 6.290 CEGUI::TabControl Class Reference

Base class for standard Tab Control widget.

Inheritance diagram for CEGUI::TabControl:



Collaboration diagram for CEGUI::TabControl:



### Public Types

- enum **TabPanePosition** { **Top**, **Bottom** }

### Public Member Functions

- size\_t **getTabCount** (void) const  
*Return number of tabs.*
- TabPanePosition **getTabPanePosition** (void) const  
*Return the positioning of the tab pane.*
- void **setTabPanePosition** (TabPanePosition pos)  
*Change the positioning of the tab button pane.*
- void **setSelectedTab** (const String &name)  
*Set the selected tab by the name of the root window within it. Also ensures that the tab is made visible (tab pane is scrolled if required).*  
**Exceptions:**  
*InvalidRequestException* thrown if there's no tab named name.
- void **setSelectedTab** (uint ID)



Set the selected tab by the ID of the root window within it. Also ensures that the tab is made visible (tab pane is scrolled if required).

**Exceptions:**

***InvalidRequestException** thrown if index is out of range.*

- void **setSelectedTabAtIndex** (size\_t index)

Set the selected tab by the index position in the tab control. Also ensures that the tab is made visible (tab pane is scrolled if required).

**Exceptions:**

***InvalidRequestException** thrown if index is out of range.*

- void **makeTabVisible** (const **String** &name)

Ensure that the tab by the name of the root window within it is visible.

**Exceptions:**

***InvalidRequestException** thrown if there's no tab named name.*

- void **makeTabVisible** (uint ID)

Ensure that the tab by the ID of the root window within it is visible.

**Exceptions:**

***InvalidRequestException** thrown if index is out of range.*

- void **makeTabVisibleAtIndex** (size\_t index)

Ensure that the tab by the index position in the tab control is visible.

**Exceptions:**

***InvalidRequestException** thrown if index is out of range.*

- **Window** \* **getTabContentsAtIndex** (size\_t index) const

Return the **Window** which is the first child of the tab at index position index.

- **Window** \* **getTabContents** (const **String** &name) const

Return the **Window** which is the tab content with the given name.

- **Window** \* **getTabContents** (uint ID) const

Return the **Window** which is the tab content with the given ID.

- bool **isTabContentsSelected** (**Window** \*wnd) const

Return whether the tab contents window is currently selected.

- size\_t **getSelectedTabIndex** () const

Return the index of the currently selected tab.

- const **UDim** & **getTabHeight** (void) const

Return the height of the tabs.

- const **UDim** & **getTabTextPadding** (void) const

Return the amount of padding to add either side of the text in the tab.

- virtual void **initialiseComponents** (void)

Initialise the [Window](#) based object ready for use.

- void [setTabHeight](#) (const [UDim](#) &height)  
*Set the height of the tabs.*
- void [setTabTextPadding](#) (const [UDim](#) &padding)  
*Set the amount of padding to add either side of the text in the tab.*
- void [addTab](#) ([Window](#) \*wnd)  
*Add a new tab to the tab control.*
- void [removeTab](#) (const [String](#) &name)  
*Remove the named tab from the tab control.*
- void [removeTab](#) (uint ID)  
*Remove the tab with the given ID from the tab control.*
- [TabControl](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [TabControl](#) base class.*
- virtual [~TabControl](#) (void)  
*Destructor for [Listbox](#) base class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventSelectionChanged](#)  
*[Event](#) triggered when there is a change to the currently selected tab.*
- static const [String](#) [ContentPaneNameSuffix](#)  
*Widget name suffix for the tab content pane component.*
- static const [String](#) [TabButtonNameSuffix](#)  
*Widget name suffix for the tab button components.*
- static const [String](#) [TabButtonPaneNameSuffix](#)  
*Widget name suffix for the tab button pane component.*
- static const [String](#) [ButtonScrollLeftSuffix](#)  
*Widget name suffix for the scroll tabs to right pane component.*
- static const [String](#) [ButtonScrollRightSuffix](#)  
*Widget name suffix for the scroll tabs to left pane component.*

## Protected Types

- typedef std::vector< [TabButton](#) \* > **TabButtonVector**

## Protected Member Functions

- virtual void [drawSelf](#) (float z)  
*Perform the actual rendering for this [Window](#).*
- virtual void [addButtonForTabContent](#) ([Window](#) \*wnd)  
*Add a [TabButton](#) for the specified child [Window](#).*
- virtual void [removeButtonForTabContent](#) ([Window](#) \*wnd)  
*Remove the [TabButton](#) for the specified child [Window](#).*
- [TabButton](#) \* [getButtonForTabContents](#) ([Window](#) \*wnd) const  
*Return the [TabButton](#) associated with this [Window](#).*  
**Exceptions:**  
*[InvalidRequestException](#) thrown if content is not found.*
- [String](#) [makeButtonName](#) ([Window](#) \*wnd)  
*Construct a button name to handle a window.*
- virtual void [selectTab\\_impl](#) ([Window](#) \*wnd)  
*Internal implementation of select tab.*
- virtual void [makeTabVisible\\_impl](#) ([Window](#) \*wnd)  
*Internal implementation of make tab visible.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- [Window](#) \* [getTabButtonPane](#) () const  
*Return a pointer to the tab button pane ([Window](#)) for this [TabControl](#).*
- [Window](#) \* [getTabPage](#) () const  
*Return a pointer to the content component widget for this [TabControl](#).*
- void [performChildWindowLayout](#) ()  
*method called to perform extended laying out of attached child windows.*
- int [writeChildWindowsXML](#) ([XMLSerializer](#) &xml\_stream) const
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- [TabButton](#) \* [createTabButton](#) (const [String](#) &name) const  
*create and return a pointer to a [TabButton](#) widget for use as a clickable tab header*
- virtual void [onSelectionChanged](#) ([WindowEventArgs](#) &e)

*Handler called internally when the currently selected item or items changes.*

- virtual void **onFontChanged** ([WindowEventArgs](#) &e)  
*Handler called when the window's font is changed.*
- void **calculateTabButtonSizePosition** (size\_t index)  
*create and return a pointer to a [TabButton](#) widget for use as a clickable tab header*
- void **addTabControlProperties** (void)
- void **addChild\_impl** ([Window](#) \*wnd)  
*Add given window to child list at an appropriate position.*
- void **removeChild\_impl** ([Window](#) \*wnd)  
*Remove given window from child list.*
- bool **handleContentWindowTextChanged** (const [EventArgs](#) &args)
- bool **handleTabButtonClicked** (const [EventArgs](#) &args)
- bool **handleScrollPane** (const [EventArgs](#) &e)
- bool **handleDraggedPane** (const [EventArgs](#) &e)
- bool **handleWheeledPane** (const [EventArgs](#) &e)

## Protected Attributes

- [UDim](#) **d\_tabHeight**  
*The height of the tabs in pixels.*
- [UDim](#) **d\_tabPadding**  
*The padding of the tabs relative to parent.*
- [TabButtonVector](#) **d\_tabButtonVector**  
*Sorting for tabs.*
- float **d\_firstTabOffset**  
*The offset in pixels of the first tab.*
- [TabPagePosition](#) **d\_tabPanePos**  
*The position of the tab pane.*
- float **d\_btGrabPos**  
*The position on the button tab where user grabbed.*

## Static Protected Attributes

- static [TabControlProperties::TabHeight](#) **d\_tabHeightProperty**
- static [TabControlProperties::TabTextPadding](#) **d\_tabTextPaddingProperty**
- static [TabControlProperties::TabPagePosition](#) **d\_tabPanePosition**

## 6.290.1 Detailed Description

Base class for standard Tab Control widget.

## 6.290.2 Member Function Documentation

### 6.290.2.1 `size_t CEGUI::TabControl::getTabCount (void) const`

Return number of tabs.

#### Returns:

the number of tabs currently present.

### 6.290.2.2 `TabPanePosition CEGUI::TabControl::getTabPanePosition (void) const` [inline]

Return the positioning of the tab pane.

#### Returns:

The positioning of the tab window within the tab control.

### 6.290.2.3 `void CEGUI::TabControl::setTabPanePosition (TabPanePosition pos)`

Change the positioning of the tab button pane.

#### Parameters:

*pos* The new positioning of the tab pane

### 6.290.2.4 `Window * CEGUI::TabControl::getTabContentsAtIndex (size_t index) const`

Return the [Window](#) which is the first child of the tab at index position *index*.

#### Parameters:

*index* Zero based index of the item to be returned.

#### Returns:

Pointer to the [Window](#) at index position *index* in the tab control.

#### Exceptions:

[InvalidRequestException](#) thrown if *index* is out of range.

**6.290.2.5 Window \* CEGUI::TabControl::getTabContents (const String & name) const**

Return the [Window](#) which is the tab content with the given name.

**Parameters:**

*name* Name of the [Window](#) which was attached as a tab content.

**Returns:**

Pointer to the named [Window](#) in the tab control.

**Exceptions:**

[InvalidRequestException](#) thrown if content is not found.

**6.290.2.6 Window \* CEGUI::TabControl::getTabContents (uint ID) const**

Return the [Window](#) which is the tab content with the given ID.

**Parameters:**

*ID* ID of the [Window](#) which was attached as a tab content.

**Returns:**

Pointer to the [Window](#) with the given ID in the tab control.

**Exceptions:**

[InvalidRequestException](#) thrown if content is not found.

**6.290.2.7 bool CEGUI::TabControl::isTabContentsSelected (Window \* wnd) const**

Return whether the tab contents window is currently selected.

**Parameters:**

*wnd* The tab contents window to query.

**Returns:**

true if the tab is currently selected, false otherwise.

**Exceptions:**

[InvalidRequestException](#) thrown if *wnd* is not a valid tab contents window.

**6.290.2.8 size\_t CEGUI::TabControl::getSelectedTabIndex () const**

Return the index of the currently selected tab.

**Returns:**

index of the currently selected tab.

**6.290.2.9 void CEGUI::TabControl::initialiseComponents (void) [virtual]**

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.290.2.10 void CEGUI::TabControl::addTab (Window \* *wnd*)**

Add a new tab to the tab control.

The new tab will be added with the same text as the window passed in.

**Parameters:**

*wnd* The [Window](#) which will be placed in the content area of this new tab.

**6.290.2.11 void CEGUI::TabControl::removeTab (const String & *name*)**

Remove the named tab from the tab control.

The tab content will be destroyed.

**6.290.2.12 void CEGUI::TabControl::removeTab (uint *ID*)**

Remove the tab with the given ID from the tab control.

The tab content will be destroyed.

**6.290.2.13 virtual void CEGUI::TabControl::drawSelf (float *z*) [inline, protected, virtual]**

Perform the actual rendering for this [Window](#).

**Parameters:**

*z* float value specifying the base Z co-ordinate that should be used when rendering

**Returns:**

Nothing

Reimplemented from [CEGUI::Window](#).

**6.290.2.14** `void CEGUI::TabControl::selectTab_impl (Window * wnd)` [protected, virtual]

Internal implementation of select tab.

**Parameters:**

*wnd* Pointer to a [Window](#) which is the root of the tab content to select

**6.290.2.15** `void CEGUI::TabControl::makeTabVisible_impl (Window * wnd)` [protected, virtual]

Internal implementation of make tab visible.

**Parameters:**

*wnd* Pointer to a [Window](#) which is the root of the tab content to make visible

**6.290.2.16** `virtual bool CEGUI::TabControl::testClassName_impl (const String & class_name) const` [inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.290.2.17** `Window * CEGUI::TabControl::getTabButtonPane () const` [protected]

Return a pointer to the tab button pane ([Window](#)) for this [TabControl](#).

**Returns:**

Pointer to a [Window](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the component does not exist.

**6.290.2.18** `Window * CEGUI::TabControl::getTabPage () const` [protected]

Return a pointer to the content component widget for this [TabControl](#).



**Returns:**

Pointer to a [Window](#) object.

**Exceptions:**

[UnknownObjectException](#) Thrown if the component does not exist.

**6.290.2.19** `void CEGUI::TabControl::performChildWindowLayout (void)` [protected, virtual]

method called to perform extended laying out of attached child windows.

The system may call this at various times (like when it is resized for example), and it may be invoked directly where required.

**Returns:**

Nothing.

Reimplemented from [CEGUI::Window](#).

**6.290.2.20** `virtual bool CEGUI::TabControl::validateWindowRenderer (const String & name) const` [inline, protected, virtual]

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.290.2.21** `TabButton * CEGUI::TabControl::createTabButton (const String & name) const` [protected]

create and return a pointer to a [TabButton](#) widget for use as a clickable tab header

**Parameters:**

*name* Button name

**Returns:**

Pointer to a [TabButton](#) to be used for changing tabs.

**6.290.2.22** `void CEGUI::TabControl::onFontChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's font is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.290.2.23** `void CEGUI::TabControl::calculateTabButtonSizePosition (size_t index)`  
[protected]

create and return a pointer to a [TabButton](#) widget for use as a clickable tab header

**Parameters:**

*name* Button name

**Returns:**

Pointer to a [TabButton](#) to be used for changing tabs.

Calculate the correct position and size of a tab button, based on the index it is due to be placed at.

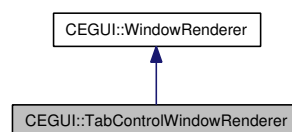
**Parameters:**

*index* The index of the tab button

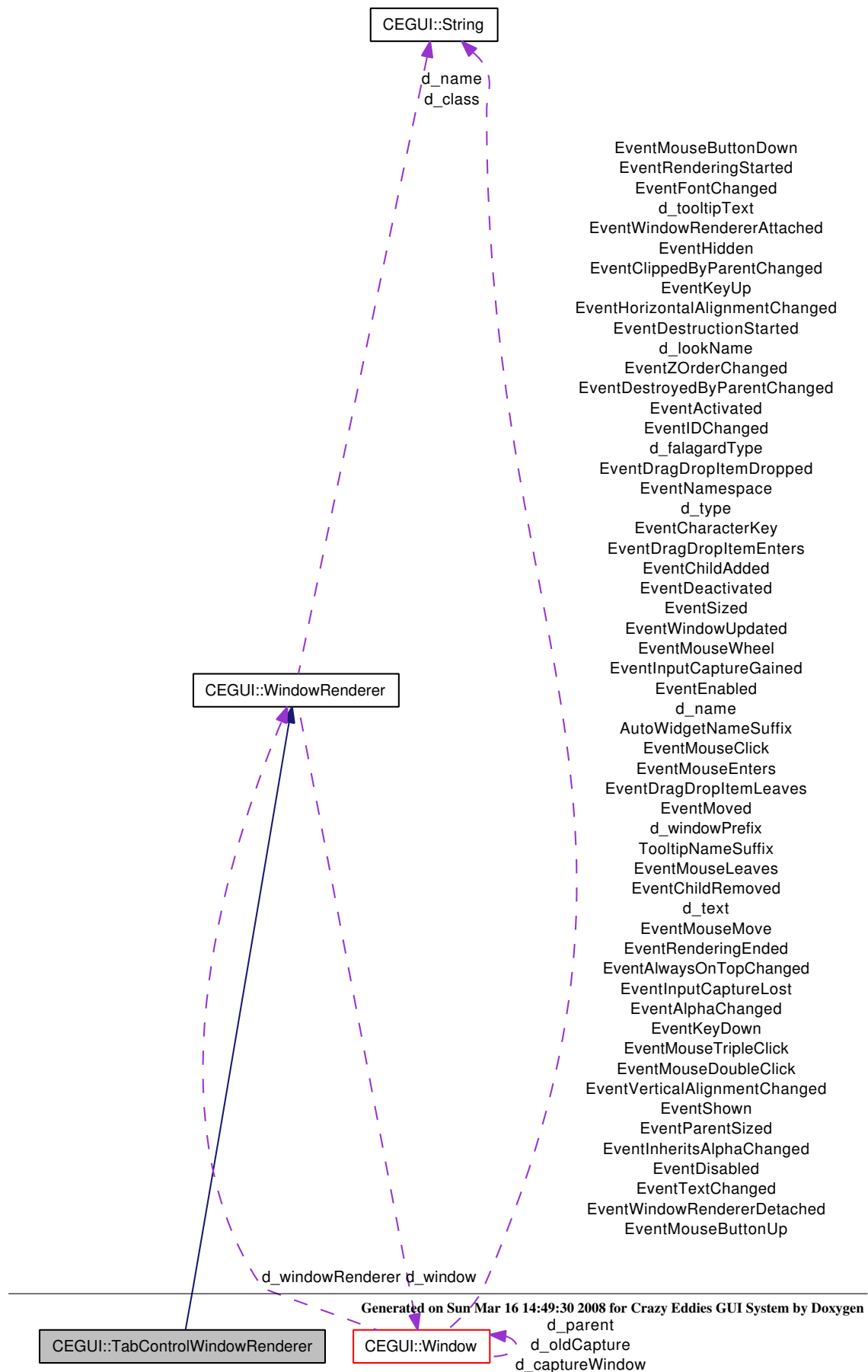
## 6.291 CEGUI::TabControlWindowRenderer Class Reference

Base class for [TabControl](#) window renderer objects.

Inheritance diagram for CEGUI::TabControlWindowRenderer:



Collaboration diagram for CEGUI::TabControlWindowRenderer:



## Public Member Functions

- [TabControlWindowRenderer](#) (const [String](#) &name)  
*Constructor.*
- virtual [TabButton](#) \* [createTabButton](#) (const [String](#) &name) const =0  
*create and return a pointer to a [TabButton](#) widget for use as a clickable tab header*

### 6.291.1 Detailed Description

Base class for [TabControl](#) window renderer objects.

### 6.291.2 Member Function Documentation

#### 6.291.2.1 virtual [TabButton](#)\* CEGUI::TabControlWindowRenderer::createTabButton (const [String](#) & *name*) const [pure virtual]

create and return a pointer to a [TabButton](#) widget for use as a clickable tab header

#### Parameters:

*name* Button name

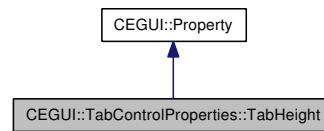
#### Returns:

Pointer to a [TabButton](#) to be used for changing tabs.

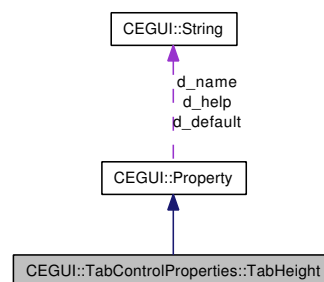
## 6.292 CEGUI::TabControlProperties::TabHeight Class Reference

[Property](#) to access the tab height setting of the tab control.

Inheritance diagram for CEGUI::TabControlProperties::TabHeight:



Collaboration diagram for CEGUI::TabControlProperties::TabHeight:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

### 6.292.1 Detailed Description

[Property](#) to access the tab height setting of the tab control.

Usage:

- Name: [TabHeight](#)
- Format: "{scale,offset}" (Unified [Dimension](#))

### 6.292.2 Member Function Documentation

#### 6.292.2.1 String CEGUI::TabControlProperties::TabHeight::get (const PropertyReceiver \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.292.2.2 void CEGUI::TabControlProperties::TabHeight::set (PropertyReceiver \* *receiver*, const String & *value*)** [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

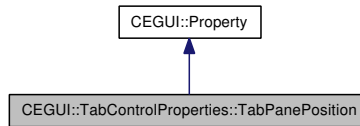
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

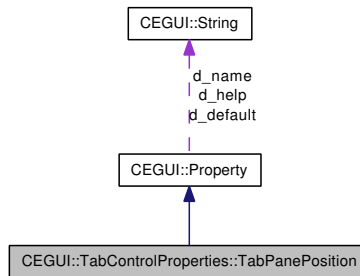
## 6.293 CEGUI::TabControlProperties::TabPanePosition Class Reference

[Property](#) to query/set the position of the button pane in tab control.

Inheritance diagram for CEGUI::TabControlProperties::TabPanePosition:



Collaboration diagram for CEGUI::TabControlProperties::TabPanePosition:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.293.1 Detailed Description

[Property](#) to query/set the position of the button pane in tab control.

Usage:

- Name: [TabPanePosition](#)
- Format: "top" | "bottom"

### 6.293.2 Member Function Documentation

#### 6.293.2.1 String CEGUI::TabControlProperties::TabPanePosition::get (const [PropertyReceiver](#) \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).



**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.293.2.2** `void CEGUI::TabControlProperties::TabPanePosition::set (PropertyReceiver * receiver,  
const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

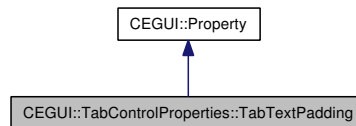
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

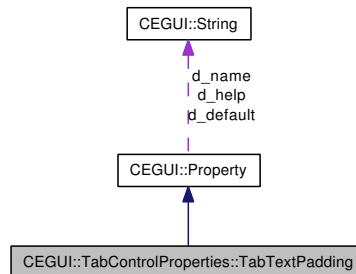
## 6.294 CEGUI::TabControlProperties::TabTextPadding Class Reference

[Property](#) to access the tab text padding setting of the tab control.

Inheritance diagram for CEGUI::TabControlProperties::TabTextPadding:



Collaboration diagram for CEGUI::TabControlProperties::TabTextPadding:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.294.1 Detailed Description

[Property](#) to access the tab text padding setting of the tab control.

Usage:

- Name: [TabTextPadding](#)
- Format: "{scale,offset}" (Unified [Dimension](#))

### 6.294.2 Member Function Documentation

#### 6.294.2.1 String CEGUI::TabControlProperties::TabTextPadding::get (const PropertyReceiver \*receiver) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

**6.294.2.2** `void CEGUI::TabControlProperties::TabTextPadding::set (PropertyReceiver * receiver,  
const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

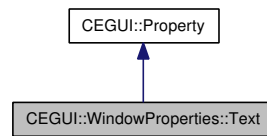
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

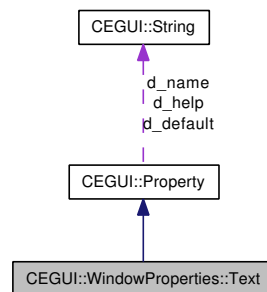
## 6.295 CEGUI::WindowProperties::Text Class Reference

[Property](#) to access window text setting.

Inheritance diagram for CEGUI::WindowProperties::Text:



Collaboration diagram for CEGUI::WindowProperties::Text:



### Public Member Functions

- [String](#) **get** (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void **set** ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.295.1 Detailed Description

[Property](#) to access window text setting.

This property offers access to the current text for the window.

#### Usage:

- Name: [Text](#)
- Format: "[text]".

#### Where:

- [text] is the name of the [Font](#) to assign for this window. The [Font](#) specified must already be loaded.

## 6.295.2 Member Function Documentation

### 6.295.2.1 String CEGUI::WindowProperties::Text::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.295.2.2 void CEGUI::WindowProperties::Text::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

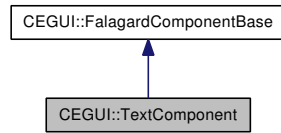
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

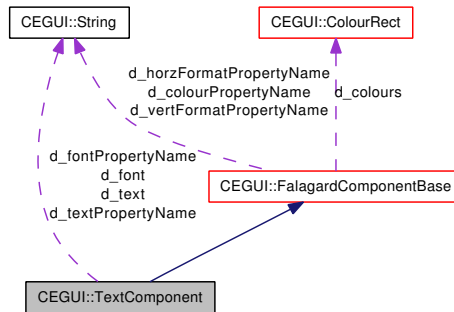
## 6.296 CEGUI::TextComponent Class Reference

Class that encapsulates information for a text component.

Inheritance diagram for CEGUI::TextComponent:



Collaboration diagram for CEGUI::TextComponent:



### Public Member Functions

- **TextComponent** ()  
*Constructor.*
- const **String** & **getText** () const  
*Return the text object that will be rendered by this **TextComponent**.*
- void **setText** (const **String** &text)  
*Set the text that will be rendered by this **TextComponent**.*
- const **String** & **getFont** () const  
*Return the name of the font to be used when rendering this **TextComponent**.*
- void **setFont** (const **String** &font)  
*Set the name of the font to be used when rendering this **TextComponent**.*
- **VerticalTextFormatting** **getVerticalFormatting** () const  
*Return the current vertical formatting setting for this **TextComponent**.*
- void **setVerticalFormatting** (**VerticalTextFormatting** fmt)  
*Set the vertical formatting setting for this **TextComponent**.*
- **HorizontalTextFormatting** **getHorizontalFormatting** () const

*Return the current horizontal formatting setting for this [TextComponent](#).*

- void [setHorizontalFormatting](#) ([HorizontalTextFormatting](#) fmt)  
*Set the horizontal formatting setting for this [TextComponent](#).*
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
*Writes an xml representation of this [TextComponent](#) to out\_stream.*
- bool [isTextFetchedFromProperty](#) () const  
*Return whether this [TextComponent](#) fetches it's text string via a property on the target window.*
- const [String](#) & [getTextPropertySource](#) () const  
*Return the name of the property that will be used to determine the text string to render for this [TextComponent](#).*
- void [setTextPropertySource](#) (const [String](#) &property)  
*Set the name of the property that will be used to determine the text string to render for this [TextComponent](#).*
- bool [isFontFetchedFromProperty](#) () const  
*Return whether this [TextComponent](#) fetches it's font via a property on the target window.*
- const [String](#) & [getFontPropertySource](#) () const  
*Return the name of the property that will be used to determine the font to use for rendering the text string for this [TextComponent](#).*
- void [setFontPropertySource](#) (const [String](#) &property)  
*Set the name of the property that will be used to determine the font to use for rendering the text string of this [TextComponent](#).*

## Protected Member Functions

- void [render\\_impl](#) ([Window](#) &srcWindow, [Rect](#) &destRect, float base\_z, const [CEGUI::ColourRect](#) \*modColours, const [Rect](#) \*clipper, bool clipToDisplay) const  
*Method to do main render caching work.*

## 6.296.1 Detailed Description

Class that encapsulates information for a text component.

## 6.296.2 Member Function Documentation

### 6.296.2.1 const [String](#) & CEGUI::TextComponent::getText () const

Return the text object that will be rendered by this [TextComponent](#).

#### Returns:

[String](#) object containing the text that will be rendered.

**6.296.2.2 void CEGUI::TextComponent::setText (const String & *text*)**

Set the text that will be rendered by this [TextComponent](#).

Note that setting this to the empty string ("") will cause the text from the base window passed when rendering to be used instead.

**Parameters:**

*text* [String](#) containing text to render, or "" to render text from window.

**Returns:**

Nothing.

**6.296.2.3 const String & CEGUI::TextComponent::getFont () const**

Return the name of the font to be used when rendering this [TextComponent](#).

**Returns:**

[String](#) object containing the name of a font, or "" if the window font is to be used.

**6.296.2.4 void CEGUI::TextComponent::setFont (const String & *font*)**

Set the name of the font to be used when rendering this [TextComponent](#).

Note that setting this to the empty string ("") will cause the font from the base window passed when rendering to be used instead.

**Parameters:**

*font* [String](#) containing name of a font

**Returns:**

Nothing.

**6.296.2.5 VerticalTextFormatting CEGUI::TextComponent::getVerticalFormatting () const**

Return the current vertical formatting setting for this [TextComponent](#).

**Returns:**

One of the VerticalTextFormatting enumerated values.

**6.296.2.6 void CEGUI::TextComponent::setVerticalFormatting (VerticalTextFormatting *fnt*)**

Set the vertical formatting setting for this [TextComponent](#).

**Parameters:**

*fnt* One of the VerticalTextFormatting enumerated values.



**Returns:**

Nothing.

**6.296.2.7 HorizontalTextFormatting CEGUI::TextComponent::getHorizontalFormatting () const**

Return the current horizontal formatting setting for this [TextComponent](#).

**Returns:**

One of the HorizontalTextFormatting enumerated values.

**6.296.2.8 void CEGUI::TextComponent::setHorizontalFormatting (HorizontalTextFormatting *fmt*)**

Set the horizontal formatting setting for this [TextComponent](#).

**Parameters:**

*fmt* One of the HorizontalTextFormatting enumerated values.

**Returns:**

Nothing.

**6.296.2.9 void CEGUI::TextComponent::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [TextComponent](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.296.2.10 bool CEGUI::TextComponent::isTextFetchedFromProperty () const**

Return whether this [TextComponent](#) fetches it's text string via a property on the target window.

**Returns:**

- true if the text string comes via a Property.
- false if the text string is defined explicitly, or will come from the target window.

**6.296.2.11   const String & CEGUI::TextComponent::getTextPropertySource () const**

Return the name of the property that will be used to determine the text string to render for this [TextComponent](#).

**Returns:**

[String](#) object holding the name of a Property.

**6.296.2.12   void CEGUI::TextComponent::setTextPropertySource (const String & *property*)**

Set the name of the property that will be used to determine the text string to render for this [TextComponent](#).

**Parameters:**

*property* [String](#) object holding the name of a Property. The property can contain any text string to render.

**Returns:**

Nothing.

**6.296.2.13   bool CEGUI::TextComponent::isFontFetchedFromProperty () const**

Return whether this [TextComponent](#) fetches it's font via a property on the target window.

**Returns:**

- true if the font comes via a Property.
- false if the font is defined explicitly, or will come from the target window.

**6.296.2.14   const String & CEGUI::TextComponent::getFontPropertySource () const**

Return the name of the property that will be used to determine the font to use for rendering the text string for this [TextComponent](#).

**Returns:**

[String](#) object holding the name of a Property.

**6.296.2.15   void CEGUI::TextComponent::setFontPropertySource (const String & *property*)**

Set the name of the property that will be used to determine the font to use for rendering the text string of this [TextComponent](#).

**Parameters:**

*property* [String](#) object holding the name of a Property. The property should access a valid font name.

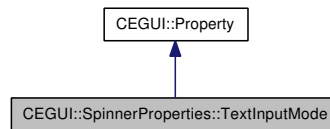
**Returns:**

Nothing.

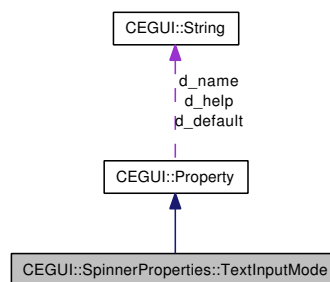
## 6.297 CEGUI::SpinnerProperties::TextInputMode Class Reference

[Property](#) to access the [TextInputMode](#) setting.

Inheritance diagram for CEGUI::SpinnerProperties::TextInputMode:



Collaboration diagram for CEGUI::SpinnerProperties::TextInputMode:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.297.1 Detailed Description

[Property](#) to access the [TextInputMode](#) setting.

This property offers access the text display and input mode for the spinner.

**Usage:**

- Name: [TextInputMode](#)
- Format: "[text]".

**Where [text] is:**

- "FloatingPoint" for floating point decimal numbers.
- "Integer" for integer decimal numbers.
- "Hexadecimal" for hexadecimal numbers.
- "Octal" for octal numbers.

## 6.297.2 Member Function Documentation

### 6.297.2.1 `String CEGUI::SpinnerProperties::TextInputMode::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.297.2.2 `void CEGUI::SpinnerProperties::TextInputMode::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

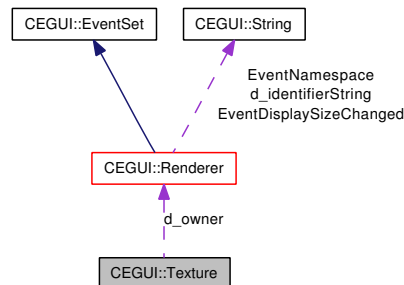
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.298 CEGUI::Texture Class Reference

Abstract base class specifying the required interface for [Texture](#) objects.

Collaboration diagram for CEGUI::Texture:



### Public Types

- enum [PixelFormat](#) { [PF\\_RGB](#), [PF\\_RGBA](#) }  
*Enum containing the list of supported pixel formats that can be passed to loadFromMemory.*

### Public Member Functions

- virtual ushort [getWidth](#) (void) const =0  
*Returns the current pixel width of the texture.*
- virtual ushort [getOriginalWidth](#) (void) const  
*Returns the original pixel width of the data loaded into the texture.*
- virtual float [getXScale](#) (void) const  
*Returns the current scale used for the width of the texture.*
- virtual ushort [getHeight](#) (void) const =0  
*Returns the current pixel height of the texture.*
- virtual ushort [getOriginalHeight](#) (void) const  
*Returns the original pixel height of the data loaded into the texture.*
- virtual float [getYScale](#) (void) const  
*Returns the current scale used for the height of the texture.*
- virtual void [loadFromFile](#) (const [String](#) &filename, const [String](#) &resourceGroup)=0  
*Loads the specified image file into the texture. The texture is resized as required to hold the image.*
- virtual void [loadFromMemory](#) (const void \*buffPtr, uint buffWidth, uint buffHeight, [PixelFormat](#) pixelFormat)=0  
*Loads (copies) an image in memory into the texture. The texture is resized as required to hold the image.*

- [Renderer](#) \* [getRenderer](#) (void) const

*Return a pointer to the [Renderer](#) object that created and owns this [Texture](#).*

## Protected Member Functions

- [Texture](#) ([Renderer](#) \*owner)

*Constructor for [Texture](#) base class. This is never called by client code.*

- virtual ~[Texture](#) (void)

*Destructor for [Texture](#) base class. This is never called by client code.*

### 6.298.1 Detailed Description

Abstract base class specifying the required interface for [Texture](#) objects.

[Texture](#) objects are created via the [Renderer](#). The actual inner workings of any [Texture](#) object are dependant upon the [Renderer](#) (and underlying API) in use. This base class defines the minimal set of functions that is required for the rest of the system to work. [Texture](#) objects are only created through the [Renderer](#) object's texture creation functions.

### 6.298.2 Member Enumeration Documentation

#### 6.298.2.1 enum CEGUI::Texture::PixelFormat

Enum containing the list of supported pixel formats that can be passed to loadFromMemory.

**Enumerator:**

**PF\_RGB** Each pixel is 3 bytes. RGB in that order.

**PF\_RGBA** Each pixel is 4 bytes. RGBA in that order.

### 6.298.3 Member Function Documentation

#### 6.298.3.1 virtual ushort CEGUI::Texture::getWidth (void) const [pure virtual]

Returns the current pixel width of the texture.

**Returns:**

ushort value that is the current width of the texture in pixels

#### 6.298.3.2 virtual ushort CEGUI::Texture::getOriginalWidth (void) const [inline, virtual]

Returns the original pixel width of the data loaded into the texture.

**Returns:**

ushort value that is the original width, in pixels, of the data last loaded into the texture.

**Note:**

for compatibility reason this method is optional the auto scale issue mantis ticket # 0000045 is not fixed for renderer that do not handle this.

**6.298.3.3 virtual float CEGUI::Texture::getXScale (void) const [inline, virtual]**

Returns the current scale used for the width of the texture.

**Returns:**

float value that denotes the horizontal scaling required to accurately map pixel positions to texture co-ords.

**6.298.3.4 virtual ushort CEGUI::Texture::getHeight (void) const [pure virtual]**

Returns the current pixel height of the texture.

**Returns:**

ushort value that is the current height of the texture in pixels

**6.298.3.5 virtual ushort CEGUI::Texture::getOriginalHeight (void) const [inline, virtual]**

Returns the original pixel height of the data loaded into the texture.

**Returns:**

ushort value that is the original height, in pixels, of the data last loaded into the texture.

**Note:**

for compatibility reason this method is optional the auto scale issue mantis ticket # 0000045 is not fixed for renderer that do not handle this.

**6.298.3.6 virtual float CEGUI::Texture::getYScale (void) const [inline, virtual]**

Returns the current scale used for the height of the texture.

**Returns:**

float value that denotes the vertical scaling required to accurately map pixel positions to texture co-ords.

**6.298.3.7** `virtual void CEGUI::Texture::loadFromFile (const String & filename, const String & resourceGroup)` [pure virtual]

Loads the specified image file into the texture. The texture is resized as required to hold the image.

**Parameters:**

*filename* The filename of the image file that is to be loaded into the texture

*resourceGroup* Resource group identifier to be passed to the resource provider when loading the image file.

**Returns:**

Nothing.

**6.298.3.8** `virtual void CEGUI::Texture::loadFromMemory (const void * buffPtr, uint buffWidth, uint buffHeight, PixelFormat pixelFormat)` [pure virtual]

Loads (copies) an image in memory into the texture. The texture is resized as required to hold the image.

**Parameters:**

*buffPtr* Pointer to the buffer containing the image data

*buffWidth* Width of the buffer (in pixels as specified by *pixelFormat* )

*buffHeight* Height of the buffer (in pixels as specified by *pixelFormat* )

*pixelFormat* PixelFormat value describing the format contained in *buffPtr*

**Returns:**

Nothing.

**6.298.3.9** `Renderer* CEGUI::Texture::getRenderer (void) const` [inline]

Return a pointer to the [Renderer](#) object that created and owns this [Texture](#).

**Returns:**

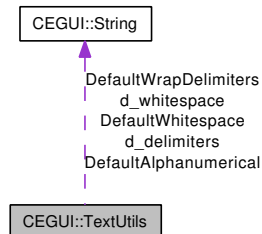
Pointer to the [Renderer](#) object that owns the [Texture](#)



## 6.299 CEGUI::TextUtils Class Reference

Text utility support class. This class is all static members. You do not create instances of this class.

Collaboration diagram for CEGUI::TextUtils:



### Static Public Member Functions

- static [String getNextWord](#) (const [String](#) &str, [String::size\\_type](#) start\_idx=0, const [String](#) &delimiters=[DefaultWhitespace](#))  
*return a [String](#) containing the the next word in a [String](#).*
- static [String::size\\_type](#) [getWordStartIdx](#) (const [String](#) &str, [String::size\\_type](#) idx)  
*Return the index of the first character of the word at idx.*
- static [String::size\\_type](#) [getNextWordStartIdx](#) (const [String](#) &str, [String::size\\_type](#) idx)  
*Return the index of the first character of the word after the word at idx.*
- static void [trimLeadingChars](#) ([String](#) &str, const [String](#) &chars)  
*Trim all characters from the set specified in chars from the begining of str.*
- static void [trimTrailingChars](#) ([String](#) &str, const [String](#) &chars)  
*Trim all characters from the set specified in chars from the end of str.*

### Static Public Attributes

- static const [String DefaultWhitespace](#)  
*The default set of whitespace.*
- static const [String DefaultAlphanumerical](#)  
*default set of alphanumericals.*
- static const [String DefaultWrapDelimiters](#)  
*The default set of word-wrap delimiters.*

#### 6.299.1 Detailed Description

Text utility support class. This class is all static members. You do not create instances of this class.

## 6.299.2 Member Function Documentation

### 6.299.2.1 String CEGUI::TextUtils::getNextWord (const String & *str*, String::size\_type *start\_idx* = 0, const String & *delimiters* = DefaultWhitespace) [static]

return a [String](#) containing the the next word in a [String](#).

This method returns a [String](#) object containing the the word, starting at index *start\_idx*, of [String](#) *str* as delimited by the code points specified in string *delimiters* (or the ends of the input string).

#### Parameters:

*str* [String](#) object containing the input data.

*start\_idx* index into *str* where the search for the next word is to begin. Defaults to start of *str*.

*delimiters* [String](#) object containing the set of delimiter code points to be used when determining the start and end points of a word in string *str*. Defaults to whitespace.

#### Returns:

[String](#) object containing the next *delimiters* delimited word from *str*, starting at index *start\_idx*.

### 6.299.2.2 String::size\_type CEGUI::TextUtils::getWordStartIdx (const String & *str*, String::size\_type *idx*) [static]

Return the index of the first character of the word at *idx*.

/note This currently uses DefaultWhitespace and DefaultAlphanumerical to determine groupings for what constitutes a 'word'.

#### Parameters:

*str* [String](#) containing text.

*idx* Index into *str* where search for start of word is to begin.

#### Returns:

Index into *str* which marks the begining of the word at index *idx*.

### 6.299.2.3 String::size\_type CEGUI::TextUtils::getNextWordStartIdx (const String & *str*, String::size\_type *idx*) [static]

Return the index of the first character of the word after the word at *idx*.

/note This currently uses DefaultWhitespace and DefaultAlphanumerical to determine groupings for what constitutes a 'word'.

#### Parameters:

*str* [String](#) containing text.

*idx* Index into *str* where search is to begin.

#### Returns:

Index into *str* which marks the begining of the word at after the word at index *idx*. If *idx* is within the last word, then the return is the last index in *str*.

**6.299.2.4 void CEGUI::TextUtils::trimLeadingChars (String & *str*, const String & *chars*)**  
[static]

Trim all characters from the set specified in *chars* from the beginning of *str*.

**Parameters:**

*str* String object to be trimmed.

*chars* String object containing the set of code points to be trimmed.

**6.299.2.5 void CEGUI::TextUtils::trimTrailingChars (String & *str*, const String & *chars*)**  
[static]

Trim all characters from the set specified in *chars* from the end of *str*.

**Parameters:**

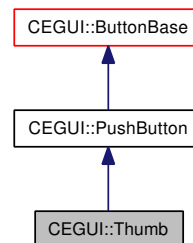
*str* String object to be trimmed.

*chars* String object containing the set of code points to be trimmed.

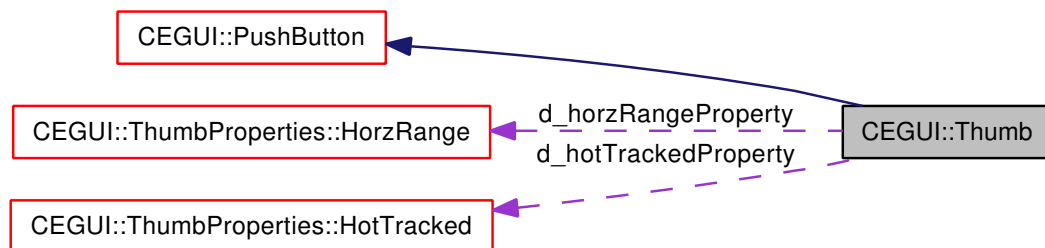
## 6.300 CEGUI::Thumb Class Reference

Base class for [Thumb](#) widget.

Inheritance diagram for CEGUI::Thumb:



Collaboration diagram for CEGUI::Thumb:



### Public Member Functions

- bool [isHotTracked](#) (void) const  
*return whether hot-tracking is enabled or not.*
- bool [isVertFree](#) (void) const  
*return whether the thumb is movable on the vertical axis.*
- bool [isHorzFree](#) (void) const  
*return whether the thumb is movable on the horizontal axis.*
- std::pair< float, float > [getVertRange](#) (void) const  
*Return a std::pair that describes the current range set for the vertical movement.*
- std::pair< float, float > [getHorzRange](#) (void) const  
*Return a std::pair that describes the current range set for the horizontal movement.*
- void [setHotTracked](#) (bool setting)  
*set whether the thumb uses hot-tracking.*
- void [setVertFree](#) (bool setting)  
*set whether thumb is movable on the vertical axis.*

- void [setHorzFree](#) (bool setting)  
*set whether thumb is movable on the horizontal axis.*
- void [setVertRange](#) (float min, float max)  
*set the movement range of the thumb for the vertical axis.*
- void [setHorzRange](#) (float min, float max)  
*set the movement range of the thumb for the horizontal axis.*
- [Thumb](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Thumb](#) objects.*
- virtual [~Thumb](#) (void)  
*Destructor for [Thumb](#) objects.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*
- static const [String](#) [EventThumbPositionChanged](#)  
*The position of the thumb widget has changed.*
- static const [String](#) [EventThumbTrackStarted](#)  
*Name of the event fired when the user begins dragging the thumb.*
- static const [String](#) [EventThumbTrackEnded](#)  
*Name of the event fired when the user releases the thumb.*

## Protected Member Functions

- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual void [onThumbPositionChanged](#) ([WindowEventArgs](#) &e)  
*event triggered internally when the position of the thumb*
- virtual void [onThumbTrackStarted](#) ([WindowEventArgs](#) &e)  
*Handler triggered when the user begins to drag the thumb.*
- virtual void [onThumbTrackEnded](#) ([WindowEventArgs](#) &e)  
*Handler triggered when the thumb is released.*

- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onMouseDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*

## Protected Attributes

- bool [d\\_hotTrack](#)  
*true if events are to be sent real-time, else just when thumb is released*
- bool [d\\_vertFree](#)  
*true if thumb is movable vertically*
- bool [d\\_horzFree](#)  
*true if thumb is movable horizontally*
- float [d\\_vertMin](#)
- float [d\\_vertMax](#)  
*vertical range*
- float [d\\_horzMin](#)
- float [d\\_horzMax](#)  
*horizontal range*
- bool [d\\_beingDragged](#)  
*true if thumb is being dragged*
- [Point](#) [d\\_dragPoint](#)  
*point where we are being dragged at.*

### 6.300.1 Detailed Description

Base class for [Thumb](#) widget.

The thumb widget is used to compose other widgets (like sliders and scroll bars). You would not normally need to use this widget directly unless you are making a new widget of some type.

### 6.300.2 Member Function Documentation

#### 6.300.2.1 `bool CEGUI::Thumb::isHotTracked (void) const` `[inline]`

return whether hot-tracking is enabled or not.

**Returns:**

true if hot-tracking is enabled. false if hot-tracking is disabled.

**6.300.2.2 bool CEGUI::Thumb::isVertFree (void) const [inline]**

return whether the thumb is movable on the vertical axis.

**Returns:**

true if the thumb is movable along the vertical axis. false if the thumb is fixed on the vertical axis.

**6.300.2.3 bool CEGUI::Thumb::isHorzFree (void) const [inline]**

return whether the thumb is movable on the horizontal axis.

**Returns:**

true if the thumb is movable along the horizontal axis. false if the thumb is fixed on the horizontal axis.

**6.300.2.4 std::pair< float, float > CEGUI::Thumb::getVertRange (void) const**

Return a std::pair that describes the current range set for the vertical movement.

**Returns:**

a std::pair describing the current vertical range. The first element is the minimum value, the second element is the maximum value.

**6.300.2.5 std::pair< float, float > CEGUI::Thumb::getHorzRange (void) const**

Return a std::pair that describes the current range set for the horizontal movement.

**Returns:**

a std::pair describing the current horizontal range. The first element is the minimum value, the second element is the maximum value.

**6.300.2.6 void CEGUI::Thumb::setHotTracked (bool *setting*) [inline]**

set whether the thumb uses hot-tracking.

**Parameters:**

*setting* true to enable hot-tracking. false to disable hot-tracking.

**Returns:**

Nothing.

**6.300.2.7 void CEGUI::Thumb::setVertFree (bool *setting*) [inline]**

set whether thumb is movable on the vertical axis.

**Parameters:**

*setting* true to allow movement of thumb along the vertical axis. false to fix thumb on the vertical axis.

**Returns:**

nothing.

**6.300.2.8 void CEGUI::Thumb::setHorzFree (bool *setting*) [inline]**

set whether thumb is movable on the horizontal axis.

**Parameters:**

*setting* true to allow movement of thumb along the horizontal axis. false to fix thumb on the horizontal axis.

**Returns:**

nothing.

**6.300.2.9 void CEGUI::Thumb::setVertRange (float *min*, float *max*)**

set the movement range of the thumb for the vertical axis.

The values specified here are relative to the parent window for the thumb, and are specified in whichever metrics mode is active for the widget.

**Parameters:**

*min* the minimum setting for the thumb on the vertical axis.

*max* the maximum setting for the thumb on the vertical axis.

**Returns:**

Nothing.

**6.300.2.10 void CEGUI::Thumb::setHorzRange (float *min*, float *max*)**

set the movement range of the thumb for the horizontal axis.

The values specified here are relative to the parent window for the thumb, and are specified in whichever metrics mode is active for the widget.

**Parameters:**

*min* the minimum setting for the thumb on the horizontal axis.

*max* the maximum setting for the thumb on the horizontal axis.

**Returns:**

Nothing.



**6.300.2.11** `virtual bool CEGUI::Thumb::testClassName_impl (const String & class_name) const`  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::PushButton](#).

**6.300.2.12** `void CEGUI::Thumb::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

**6.300.2.13** `void CEGUI::Thumb::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::ButtonBase](#).

**6.300.2.14** `void CEGUI::Thumb::onCaptureLost (WindowEventArgs & e)` [protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

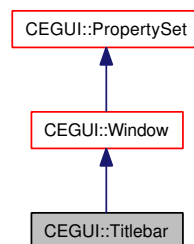
*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::ButtonBase](#).

## 6.301 CEGUI::Titlebar Class Reference

Class representing the title bar for Frame Windows.

Inheritance diagram for CEGUI::Titlebar:



Collaboration diagram for CEGUI::Titlebar:



## Public Member Functions

- bool [isDraggingEnabled](#) (void) const  
*Return whether this title bar will respond to dragging.*
- void [setDraggingEnabled](#) (bool setting)  
*Set whether this title bar widget will respond to dragging.*
- [Titlebar](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Titlebar](#) base class.*
- virtual [~Titlebar](#) (void)  
*Destructor for [Titlebar](#) base class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [WidgetTypeName](#)  
*[Window](#) factory name.*

## Protected Member Functions

- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onMouseDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void [onMouseDoubleClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been double-clicked within this window's area.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void [onFontChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's font is changed.*
- virtual void [onDraggingModeChanged](#) ([WindowEventArgs](#) &e)  
*[Event](#) handler called when the 'draggable' state for the title bar is changed.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*

## Protected Attributes

- bool [d\\_dragging](#)  
*set to true when the window is being dragged.*
- [Point](#) [d\\_dragPoint](#)  
*Point at which we are being dragged.*
- bool [d\\_dragEnabled](#)  
*true when dragging for the widget is enabled.*
- [Rect](#) [d\\_oldCursorArea](#)  
*Used to backup cursor restraint area.*

### 6.301.1 Detailed Description

Class representing the title bar for Frame Windows.

### 6.301.2 Member Function Documentation

#### 6.301.2.1 bool CEGUI::Titlebar::isDraggingEnabled (void) const

Return whether this title bar will respond to dragging.

**Returns:**

true if the title bar will respond to dragging, false if the title bar will not respond.

#### 6.301.2.2 void CEGUI::Titlebar::setDraggingEnabled (bool *setting*)

Set whether this title bar widget will respond to dragging.

**Parameters:**

*setting* true if the title bar should respond to being dragged, false if it should not respond.

**Returns:**

Nothing.

#### 6.301.2.3 void CEGUI::Titlebar::onMouseMove (MouseEventArgs & *e*) [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.301.2.4** `void CEGUI::Titlebar::onMouseButtonDown (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.301.2.5** `void CEGUI::Titlebar::onMouseButtonUp (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.301.2.6** `void CEGUI::Titlebar::onMouseDoubleClicked (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been double-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.301.2.7** `void CEGUI::Titlebar::onCaptureLost (WindowEventArgs & e)` [protected, virtual]

Handler called when this window loses capture of mouse inputs.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.301.2.8** `void CEGUI::Titlebar::onFontChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's font is changed.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.301.2.9** `virtual void CEGUI::Titlebar::onDraggingModeChanged (WindowEventArgs & e)`  
[inline, protected, virtual]

[Event](#) handler called when the 'draggable' state for the title bar is changed.

Note that this is for 'internal' use at the moment and as such does not add or fire a public [Event](#) that can be subscribed to.

**6.301.2.10** `virtual bool CEGUI::Titlebar::testClassName_impl (const String & class_name) const`  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

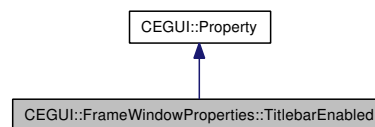
true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

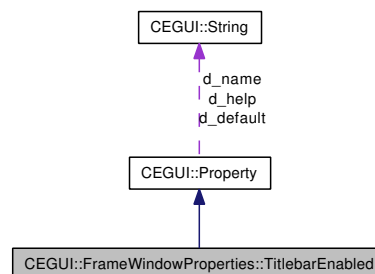
## 6.302 CEGUI::FrameWindowProperties::TitlebarEnabled Class Reference

[Property](#) to access the setting for whether the window title-bar will be enabled (or displayed depending upon choice of final widget type).

Inheritance diagram for CEGUI::FrameWindowProperties::TitlebarEnabled:



Collaboration diagram for CEGUI::FrameWindowProperties::TitlebarEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.302.1 Detailed Description

[Property](#) to access the setting for whether the window title-bar will be enabled (or displayed depending upon choice of final widget type).

#### Usage:

- Name: [TitlebarEnabled](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the windows title bar should be enabled (and/or visible)
- "False" to indicate the windows title bar should be disabled (and/or hidden)



## 6.302.2 Member Function Documentation

### 6.302.2.1 String CEGUI::FrameWindowProperties::TitlebarEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.302.2.2 void CEGUI::FrameWindowProperties::TitlebarEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

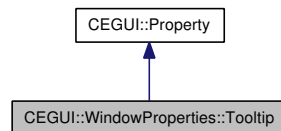
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

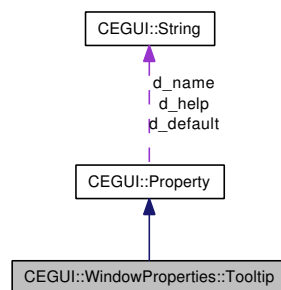
## 6.303 CEGUI::WindowProperties::Tooltip Class Reference

[Property](#) to access the tooltip text for this [Window](#).

Inheritance diagram for CEGUI::WindowProperties::Tooltip:



Collaboration diagram for CEGUI::WindowProperties::Tooltip:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.303.1 Detailed Description

[Property](#) to access the tooltip text for this [Window](#).

#### Usage:

- Name: [Tooltip](#)
- Format: "[text]".

#### Where:

- [\[Text\]](#) is the tooltip text for this window.

## 6.303.2 Member Function Documentation

### 6.303.2.1 String CEGUI::WindowProperties::Tooltip::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.303.2.2 void CEGUI::WindowProperties::Tooltip::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

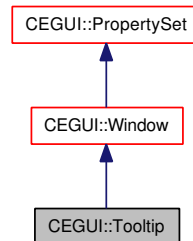
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

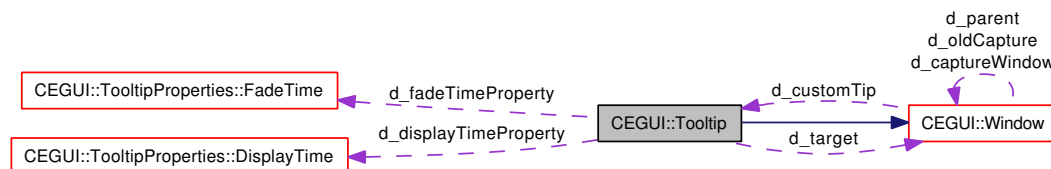
## 6.304 CEGUI::Tooltip Class Reference

Base class for [Tooltip](#) widgets.

Inheritance diagram for CEGUI::Tooltip:



Collaboration diagram for CEGUI::Tooltip:



### Public Member Functions

- [Tooltip](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for the [Tooltip](#) base class constructor.*
- [~Tooltip](#) (void)  
*Destructor for the [Tooltip](#) base class.*
- void [setTargetWindow](#) ([Window](#) \*wnd)  
*Sets the target window for the tooltip. This used internally to manage tooltips, you should not have to call this yourself.*
- const [Window](#) \* [getTargetWindow](#) ()  
*return the current target window for this [Tooltip](#).*
- void [resetTimer](#) (void)  
*Resets the timer on the tooltip when in the Active / Inactive states. This is used internally to control the tooltip, it is not normally necessary to call this method yourself.*
- float [getHoverTime](#) (void) const  
*Return the number of seconds the mouse should hover stationary over the target window before the tooltip gets activated.*
- void [setDisplayTime](#) (float seconds)  
*Set the number of seconds the tooltip should be displayed for before it automatically de-activates itself. 0 indicates that the tooltip should never timeout and auto-deactivate.*

- float [getFadeTime](#) (void) const  
*Return the number of seconds that should be taken to fade the tooltip into and out of visibility.*
- void [setHoverTime](#) (float seconds)  
*Set the number of seconds the mouse should hover stationary over the target window before the tooltip gets activated.*
- float [getDisplayTime](#) (void) const  
*Return the number of seconds the tooltip should be displayed for before it automatically de-activates itself. 0 indicates that the tooltip never timesout and auto-deactivates.*
- void [setFadeTime](#) (float seconds)  
*Set the number of seconds that should be taken to fade the tooltip into and out of visibility.*
- void [positionSelf](#) (void)  
*Causes the tooltip to position itself appropriately.*
- void [sizeSelf](#) (void)  
*Causes the tooltip to resize itself appropriately.*
- [Size](#) [getTextSize](#) () const  
*Return the size of the area that will be occupied by the tooltip text, given any current formatting options.*
- virtual [Size](#) [getTextSize\\_impl](#) () const  
*Return the size of the area that will be occupied by the tooltip text, given any current formatting options.*

## Static Public Attributes

- static const [String](#) [WidgetTypeName](#)  
*Window factory name.*
- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) [EventHoverTimeChanged](#)  
*Event fired when the hover timeout gets changed.*
- static const [String](#) [EventDisplayTimeChanged](#)  
*Event fired when the display timeout gets changed.*
- static const [String](#) [EventFadeTimeChanged](#)  
*Event fired when the fade timeout gets changed.*
- static const [String](#) [EventTooltipActive](#)  
*Event fired when the tooltip is about to get activated.*
- static const [String](#) [EventTooltipInactive](#)  
*Event fired when the tooltip has been deactivated.*

## Protected Types

- enum `TipState` { `Inactive`, `Active`, `FadeIn`, `FadeOut` }

*states for tooltip*

## Protected Member Functions

- void `doActiveState` (float elapsed)
- void `doInactiveState` (float elapsed)
- void `doFadeInState` (float elapsed)
- void `doFadeOutState` (float elapsed)
- void `switchToInactiveState` (void)
- void `switchToActiveState` (void)
- void `switchToFadeInState` (void)
- void `switchToFadeOutState` (void)
- virtual bool `testClassName_impl` (const `String` &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- virtual bool `validateWindowRenderer` (const `String` &name) const  
*Function used in checking if a `WindowRenderer` is valid for this window.*
- virtual void `onHoverTimeChanged` (`WindowEventArgs` &e)  
*`Event` trigger method called when the hover timeout gets changed.*
- virtual void `onDisplayTimeChanged` (`WindowEventArgs` &e)  
*`Event` trigger method called when the display timeout gets changed.*
- virtual void `onFadeTimeChanged` (`WindowEventArgs` &e)  
*`Event` trigger method called when the fade timeout gets changed.*
- virtual void `onTooltipActive` (`WindowEventArgs` &e)  
*`Event` trigger method called just before the tooltip becomes active.*
- virtual void `onTooltipInactive` (`WindowEventArgs` &e)  
*`Event` trigger method called just after the tooltip is deactivated.*
- void `updateSelf` (float elapsed)  
*Perform actual update processing for this `Window`.*
- void `onMouseEnters` (`MouseEventArgs` &e)  
*Handler called when the mouse cursor has entered this window's area.*
- void `onTextChanged` (`WindowEventArgs` &e)  
*Handler called when the window's text is changed.*

## Protected Attributes

- [TipState d\\_state](#)  
*Current tooltip state.*
- float [d\\_elapsed](#)  
*Used to track state change timings.*
- const [Window \\* d\\_target](#)  
*Current target [Window](#) for this [Tooltip](#).*
- float [d\\_hoverTime](#)  
*tool-tip hover time (seconds mouse must stay stationary before tip shows).*
- float [d\\_displayTime](#)  
*tool-tip display time (seconds that tip is shown for).*
- float [d\\_fadeTime](#)  
*tool-tip fade time (seconds it takes for tip to fade in and/or out).*

### 6.304.1 Detailed Description

Base class for [Tooltip](#) widgets.

The [Tooltip](#) class shows a simple pop-up window around the mouse position with some text information. The tool-tip fades in when the user hovers with the mouse over a window which has tool-tip text set, and then fades out after some pre-set time.

#### Note:

For [Tooltip](#) to work properly, you must specify a default tool-tip widget type via [System::setTooltip](#), or by setting a custom tool-tip object for your [Window\(s\)](#). Additionally, you need to ensure that time pulses are properly passed to the system via [System::injectTimePulse](#).

### 6.304.2 Member Enumeration Documentation

#### 6.304.2.1 enum CEGUI::Tooltip::TipState [protected]

states for tooltip

#### Enumerator:

**Inactive** [Tooltip](#) is currently inactive.

**Active** [Tooltip](#) is currently displayed and active.

**FadeIn** [Tooltip](#) is currently transitioning from Inactive to Active state.

**FadeOut** [Tooltip](#) is currently transitioning from Active to Inactive state.

### 6.304.3 Member Function Documentation

#### 6.304.3.1 void CEGUI::Tooltip::setTargetWindow (Window \* *wnd*)

Sets the target window for the tooltip. This used internally to manage tooltips, you should not have to call this yourself.

**Parameters:**

*wnd* [Window](#) object that the tooltip should be associated with (for now).

**Returns:**

Nothing.

#### 6.304.3.2 const Window \* CEGUI::Tooltip::getTargetWindow ()

return the current target window for this [Tooltip](#).

**Returns:**

Pointer to the target window for this [Tooltip](#) or 0 for none.

#### 6.304.3.3 void CEGUI::Tooltip::resetTimer (void)

Resets the timer on the tooltip when in the Active / Inactive states. This is used internally to control the tooltip, it is not normally necessary to call this method yourself.

**Returns:**

Nothing.

#### 6.304.3.4 float CEGUI::Tooltip::getHoverTime (void) const

Return the number of seconds the mouse should hover stationary over the target window before the tooltip gets activated.

**Returns:**

float value representing a number of seconds.

#### 6.304.3.5 void CEGUI::Tooltip::setDisplayTime (float *seconds*)

Set the number of seconds the tooltip should be displayed for before it automatically de-activates itself. 0 indicates that the tooltip should never timeout and auto-deactivate.

**Parameters:**

*seconds* float value representing a number of seconds.

**Returns:**

Nothing.



**6.304.3.6 float CEGUI::Tooltip::getFadeTime (void) const**

Return the number of seconds that should be taken to fade the tooltip into and out of visibility.

**Returns:**

float value representing a number of seconds.

**6.304.3.7 void CEGUI::Tooltip::setHoverTime (float *seconds*)**

Set the number of seconds the mouse should hover stationary over the target window before the tooltip gets activated.

**Parameters:**

*seconds* float value representing a number of seconds.

**Returns:**

Nothing.

**6.304.3.8 float CEGUI::Tooltip::getDisplayTime (void) const**

Return the number of seconds the tooltip should be displayed for before it automatically de-activates itself. 0 indicates that the tooltip never timesout and auto-deactivates.

**Returns:**

float value representing a number of seconds.

**6.304.3.9 void CEGUI::Tooltip::setFadeTime (float *seconds*)**

Set the number of seconds that should be taken to fade the tooltip into and out of visibility.

**Parameters:**

*seconds* float value representing a number of seconds.

**Returns:**

Nothing.

**6.304.3.10 void CEGUI::Tooltip::positionSelf (void)**

Causes the tooltip to position itself appropriately.

**Returns:**

Nothing.

**6.304.3.11 void CEGUI::Tooltip::sizeSelf (void)**

Causes the tooltip to resize itself appropriately.

**Returns:**

Nothing.

**6.304.3.12 Size CEGUI::Tooltip::getTextSize () const**

Return the size of the area that will be occupied by the tooltip text, given any current formatting options.

**Returns:**

[Size](#) object describing the size of the rendered tooltip text in pixels.

**6.304.3.13 Size CEGUI::Tooltip::getTextSize\_impl () const [virtual]**

Return the size of the area that will be occupied by the tooltip text, given any current formatting options.

**Returns:**

[Size](#) object describing the size of the rendered tooltip text in pixels.

**6.304.3.14 virtual bool CEGUI::Tooltip::testClassName\_impl (const String & class\_name) const [inline, protected, virtual]**

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.304.3.15 virtual bool CEGUI::Tooltip::validateWindowRenderer (const String & name) const [inline, protected, virtual]**

Function used in checking if a [WindowRenderer](#) is valid for this window.

**Returns:**

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented from [CEGUI::Window](#).

**6.304.3.16** void CEGUI::Tooltip::onHoverTimeChanged (WindowEventArgs & *e*)  
[protected, virtual]

[Event](#) trigger method called when the hover timeout gets changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.304.3.17** void CEGUI::Tooltip::onDisplayTimeChanged (WindowEventArgs & *e*)  
[protected, virtual]

[Event](#) trigger method called when the display timeout gets changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.304.3.18** void CEGUI::Tooltip::onFadeTimeChanged (WindowEventArgs & *e*) [protected, virtual]

[Event](#) trigger method called when the fade timeout gets changed.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.304.3.19** void CEGUI::Tooltip::onTooltipActive (WindowEventArgs & *e*) [protected, virtual]

[Event](#) trigger method called just before the tooltip becomes active.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.304.3.20** `void CEGUI::Tooltip::onTooltipInactive (WindowEventArgs & e)` [protected, virtual]

[Event](#) trigger method called just after the tooltip is deactivated.

**Parameters:**

*e* [WindowEventArgs](#) object.

**Returns:**

Nothing.

**6.304.3.21** `void CEGUI::Tooltip::updateSelf (float elapsed)` [protected, virtual]

Perform actual update processing for this [Window](#).

**Parameters:**

*elapsed* float value indicating the number of seconds elapsed since the last update call.

**Returns:**

Nothing.

Reimplemented from [CEGUI::Window](#).

**6.304.3.22** `void CEGUI::Tooltip::onMouseEnters (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has entered this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.304.3.23** `void CEGUI::Tooltip::onTextChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

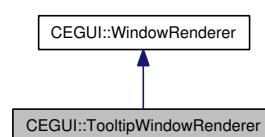
*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

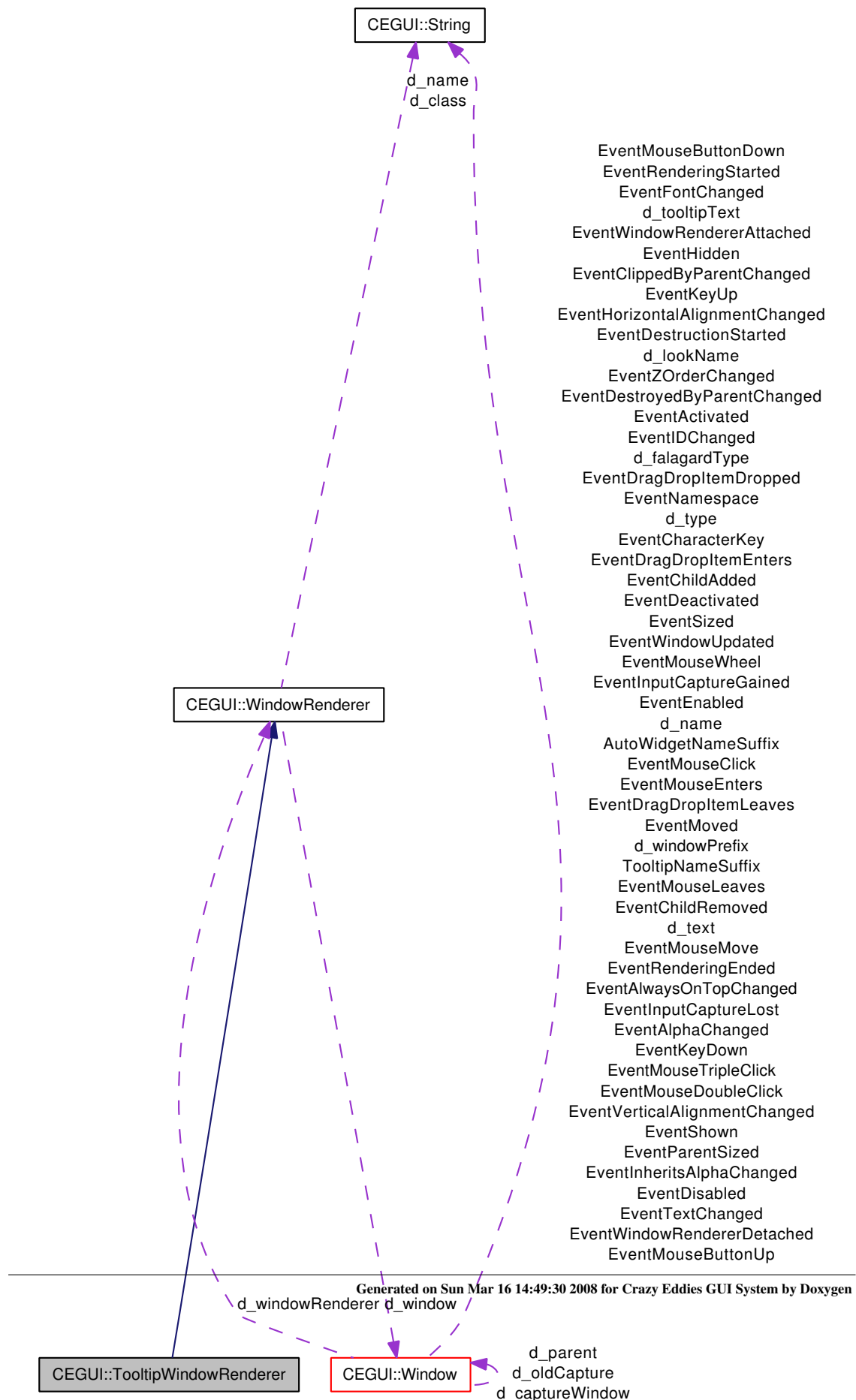
## 6.305 CEGUI::TooltipWindowRenderer Class Reference

Base class for [Tooltip](#) window renderer objects.

Inheritance diagram for CEGUI::TooltipWindowRenderer:



Collaboration diagram for CEGUI::TooltipWindowRenderer:



## Public Member Functions

- [TooltipWindowRenderer](#) (const [String](#) &name)

*Constructor.*

- virtual [Size](#) [getTextSize](#) () const =0

*Return the size of the area that will be occupied by the tooltip text, given any current formatting options.*

### 6.305.1 Detailed Description

Base class for [Tooltip](#) window renderer objects.

### 6.305.2 Member Function Documentation

#### 6.305.2.1 virtual [Size](#) CEGUI::TooltipWindowRenderer::getTextSize () const [pure virtual]

Return the size of the area that will be occupied by the tooltip text, given any current formatting options.

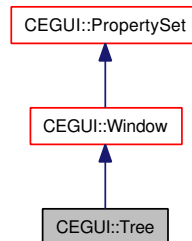
#### Returns:

[Size](#) object describing the size of the rendered tooltip text in pixels.

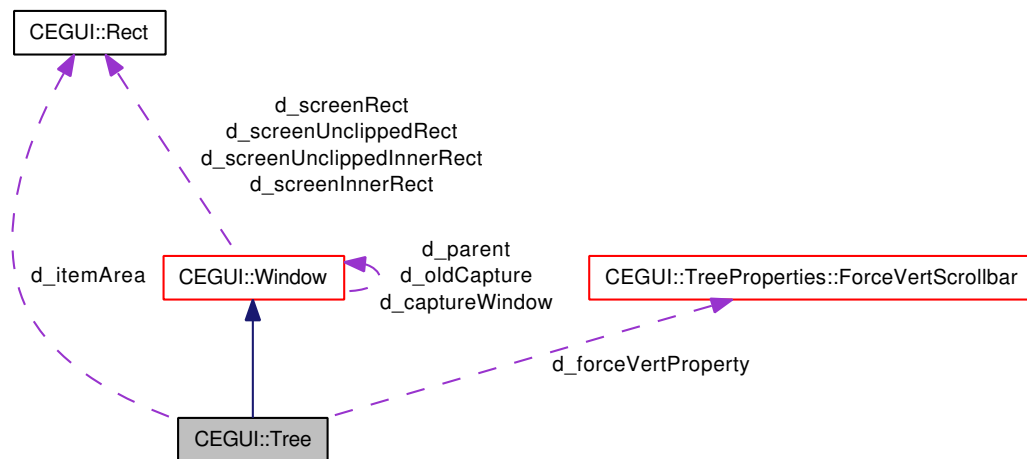
## 6.306 CEGUI::Tree Class Reference

Base class for standard [Tree](#) widget.

Inheritance diagram for CEGUI::Tree:



Collaboration diagram for CEGUI::Tree:



### Public Member Functions

- void **doTreeRender** ()
- void **doScrollbars** ()
- size\_t **getItemCount** (void) const  
*Return number of items attached to the list box.*
- size\_t **getSelectedCount** (void) const  
*Return the number of selected items in the list box.*
- [TreeItem](#) \* **getFirstSelectedItem** (void) const  
*Return a pointer to the first selected item.*
- [TreeItem](#) \* **getLastSelectedItem** (void) const  
*Return a pointer to the first selected item.*
- [TreeItem](#) \* **getNextSelected** (const [TreeItem](#) \*start\_item) const



*Return a pointer to the next selected item after item start\_item.*

- **TreeItem \* getNextSelectedItemFromList** (const LBItemList &itemList, const **TreeItem** \*start\_item, bool foundStartItem) const
- bool **isSortEnabled** (void) const

*return whether list sorting is enabled*

- void **setItemRenderArea** (**Rect** &r)
- **Scrollbar** \* **getVertScrollbar** ()
- **Scrollbar** \* **getHorzScrollbar** ()
- bool **isMultiselectEnabled** (void) const

*return whether multi-select is enabled*

- bool **isItemTooltipsEnabled** (void) const
- **TreeItem** \* **findFirstItemWithText** (const **String** &text)

*Search the list for an item with the specified text.*

- **TreeItem** \* **findNextItemWithText** (const **String** &text, const **TreeItem** \*start\_item)
- **TreeItem** \* **findItemWithTextFromList** (const LBItemList &itemList, const **String** &text, const **TreeItem** \*start\_item, bool foundStartItem)
- **TreeItem** \* **findFirstItemWithID** (uint searchID)

*Search the list for an item with the specified text.*

- **TreeItem** \* **findNextItemWithID** (uint searchID, const **TreeItem** \*start\_item)
- **TreeItem** \* **findItemWithIDFromList** (const LBItemList &itemList, uint searchID, const **TreeItem** \*start\_item, bool foundStartItem)
- bool **isTreeItemInList** (const **TreeItem** \*item) const

*Return whether the specified TreeItem is in the List.*

- bool **isVertScrollbarAlwaysShown** (void) const

*Return whether the vertical scroll bar is always shown.*

- bool **isHorzScrollbarAlwaysShown** (void) const

*Return whether the horizontal scroll bar is always shown.*

- virtual void **initialise** (void)

*Initialise the Window based object ready for use.*

- void **resetList** (void)

*Remove all items from the list.*

- void **addItem** (**TreeItem** \*item)

*Add the given TreeItem to the list.*

- void **insertItem** (**TreeItem** \*item, const **TreeItem** \*position)

*Insert an item into the list box after a specified item already in the list.*

- void **removeItem** (const **TreeItem** \*item)

*Removes the given item from the list box. If the item is has the auto delete state set, the item will be deleted.*

- void [clearAllSelections](#) (void)  
*Clear the selected state for all items.*
- bool **clearAllSelectionsFromList** (const [LBItemList](#) &itemList)
- void [setSortingEnabled](#) (bool setting)  
*Set whether the list should be sorted.*
- void [setMultiselectEnabled](#) (bool setting)  
*Set whether the list should allow multiple selections or just a single selection.*
- void [setShowVertScrollbar](#) (bool setting)  
*Set whether the vertical scroll bar should always be shown.*
- void [setShowHorzScrollbar](#) (bool setting)  
*Set whether the horizontal scroll bar should always be shown.*
- void **setItemTooltipsEnabled** (bool setting)
- void [setItemSelectState](#) ([TreeItem](#) \*item, bool state)  
*Set the select state of an attached [TreeItem](#).*
- void [setItemSelectState](#) (size\_t item\_index, bool state)  
*Set the select state of an attached [TreeItem](#).*
- virtual void [setLookNFeel](#) (const [String](#) &look)  
*Set the LookNFeel that shoule be used for this window.*
- void [handleUpdatedItemData](#) (void)  
*Causes the list box to update it's internal state after changes have been made to one or more attached [TreeItem](#) objects.*
- void [ensureItemIsVisible](#) (const [TreeItem](#) \*item)  
*Ensure the item at the specified index is visible within the list box.*
- [Tree](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Tree](#) base class.*
- virtual [~Tree](#) (void)  
*Destructor for [Tree](#) base class.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*Namespace for global events.*
- static const [String](#) **WidgetTypeName**
- static const [String](#) [EventListContentsChanged](#)  
*[Event](#) triggered when the contents of the list is changed.*
- static const [String](#) [EventSelectionChanged](#)

*Event triggered when there is a change to the currently selected item(s).*

- static const [String EventSortModeChanged](#)  
*Event triggered when the sort mode setting changes.*
- static const [String EventMultiselectModeChanged](#)  
*Event triggered when the multi-select mode setting changes.*
- static const [String EventVertScrollbarModeChanged](#)  
*Event triggered when the vertical scroll bar 'force' setting changes.*
- static const [String EventHorzScrollbarModeChanged](#)  
*Event triggered when the horizontal scroll bar 'force' setting changes.*
- static const [String EventBranchOpened](#)  
*Event triggered when a branch of the tree is opened by the user.*
- static const [String EventBranchClosed](#)  
*Event triggered when a branch of the tree is closed by the user.*

## Protected Member Functions

- virtual [Rect getTreeRenderArea](#) (void) const  
*Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.*
- virtual [Scrollbar \\*](#) [createVertScrollbar](#) (const [String](#) &name) const  
*create and return a pointer to a [Scrollbar](#) widget for use as vertical scroll bar*
- virtual [Scrollbar \\*](#) [createHorzScrollbar](#) (const [String](#) &name) const  
*create and return a pointer to a [Scrollbar](#) widget for use as horizontal scroll bar*
- virtual void [cacheTreeBaseImagery](#) ()  
*Perform caching of the widget control frame and other 'static' areas. This method should not render the actual items. Note that the items are typically rendered to layer 3, other layers can be used for rendering imagery behind and in front of the items.*
- void [addTreeEvents](#) (void)  
*Add list box specific events.*
- void [configureScrollbars](#) (void)  
*display required integrated scroll bars according to current state of the list box and update their values.*
- void [selectRange](#) (size\_t start, size\_t end)  
*select all strings between positions start and end. (inclusive) including end.*
- float [getTotalItemsHeight](#) (void) const  
*Return the sum of all item heights.*

- void **getTotalItemsInListHeight** (const LBItemList &itemList, float \*heightSum) const
- float **getWidestItemWidth** (void) const  
*Return the width of the widest item.*
- void **getWidestItemWidthInList** (const LBItemList &itemList, int itemDepth, float \*widest) const
- bool **getHeightToItemInList** (const LBItemList &itemList, const **TreeItem** \*treeItem, int itemDepth, float \*height) const  
*Clear the selected state for all items (implementation).*
- bool **clearAllSelections\_impl** (void)  
*Clear the selected state for all items (implementation).*
- **TreeItem** \* **getItemAtPoint** (const **Point** &pt) const  
*Return the **TreeItem** under the given window local pixel co-ordinate.*
- **TreeItem** \* **getItemFromListAtPoint** (const LBItemList &itemList, float \*bottomY, const **Point** &pt) const
- bool **resetList\_impl** (void)  
*Remove all items from the list.*
- virtual bool **testClassName\_impl** (const **String** &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance heirarchy.*
- bool **handle\_scrollChange** (const **EventArgs** &args)  
*Internal handler that is triggered when the user interacts with the scrollbars.*
- virtual void **populateRenderCache** ()  
*Update the rendering cache.*
- void **drawItemList** (LBItemList &itemList, **Rect** &itemsArea, float widest, **Vector3** &itemPos, **RenderCache** &cache, float alpha)
- virtual void **onListContentsChanged** (**WindowEventArgs** &e)  
*Handler called internally when the list contents are changed.*
- virtual void **onSelectionChanged** (**TreeEventArgs** &e)  
*Handler called internally when the currently selected item or items changes.*
- virtual void **onSortModeChanged** (**WindowEventArgs** &e)  
*Handler called internally when the sort mode setting changes.*
- virtual void **onMultiselectModeChanged** (**WindowEventArgs** &e)  
*Handler called internally when the multi-select mode setting changes.*
- virtual void **onVertScrollbarModeChanged** (**WindowEventArgs** &e)  
*Handler called internally when the forced display of the vertical scroll bar setting changes.*
- virtual void **onHorzScrollbarModeChanged** (**WindowEventArgs** &e)  
*Handler called internally when the forced display of the horizontal scroll bar setting changes.*

- virtual void [onBranchOpened](#) ([TreeEventArgs](#) &e)  
*Handler called internally when the user opens a branch of the tree.*
- virtual void [onBranchClosed](#) ([TreeEventArgs](#) &e)  
*Handler called internally when the user closes a branch of the tree.*
- virtual void [onSized](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's size changes.*
- virtual void [onMouseButtonDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*
- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*

## Protected Attributes

- bool [d\\_sorted](#)  
*true if list is sorted*
- bool [d\\_multiselect](#)  
*true if multi-select is enabled*
- bool [d\\_forceVertScroll](#)  
*true if vertical scrollbar should always be displayed*
- bool [d\\_forceHorzScroll](#)  
*true if horizontal scrollbar should always be displayed*
- bool [d\\_itemTooltips](#)  
*true if each item should have an individual tooltip*
- [Scrollbar](#) \* [d\\_vertScrollbar](#)  
*vertical scroll-bar widget*
- [Scrollbar](#) \* [d\\_horzScrollbar](#)  
*horizontal scroll-bar widget*
- [LBItemList](#) [d\\_listItems](#)  
*list of items in the list box.*
- [TreeItem](#) \* [d\\_lastSelected](#)  
*holds pointer to the last selected item (used in range selections)*
- [ImagerySection](#) \* [openButtonImagery](#)
- [ImagerySection](#) \* [closeButtonImagery](#)

## Friends

- class `TreeItem`

### 6.306.1 Detailed Description

Base class for standard `Tree` widget.

### 6.306.2 Member Function Documentation

#### 6.306.2.1 `size_t CEGUI::Tree::getItemCount (void) const` [inline]

Return number of items attached to the list box.

##### Returns:

the number of items currently attached to this list box.

#### 6.306.2.2 `size_t CEGUI::Tree::getSelectedCount (void) const`

Return the number of selected items in the list box.

##### Returns:

Total number of attached items that are in the selected state.

#### 6.306.2.3 `TreeItem * CEGUI::Tree::getFirstSelectedItem (void) const`

Return a pointer to the first selected item.

##### Returns:

Pointer to a `TreeItem` based object that is the first selected item in the list. will return NULL if no item is selected.

#### 6.306.2.4 `TreeItem* CEGUI::Tree::getLastSelectedItem (void) const` [inline]

Return a pointer to the first selected item.

##### Returns:

Pointer to a `TreeItem` based object that is the last item selected by the user, not necessarily the last selected in the list. Will return NULL if no item is selected.

**6.306.2.5 TreeItem \* CEGUI::Tree::getNextSelected (const TreeItem \* *start\_item*) const**

Return a pointer to the next selected item after item *start\_item*.

**Parameters:**

*start\_item* Pointer to the [TreeItem](#) where the search for the next selected item is to begin. If this parameter is NULL, the search will begin with the first item in the list box.

**Returns:**

Pointer to a [TreeItem](#) based object that is the next selected item in the list after the item specified by *start\_item*. Will return NULL if no further items were selected.

**Exceptions:**

[InvalidRequestException](#) thrown if *start\_item* is not attached to this list box.

**6.306.2.6 bool CEGUI::Tree::isSortEnabled (void) const [inline]**

return whether list sorting is enabled

**Returns:**

true if the list is sorted, false if the list is not sorted

**6.306.2.7 bool CEGUI::Tree::isMultiselectEnabled (void) const [inline]**

return whether multi-select is enabled

**Returns:**

true if multi-select is enabled, false if multi-select is not enabled.

**6.306.2.8 TreeItem \* CEGUI::Tree::findFirstItemWithText (const String & *text*)**

Search the list for an item with the specified text.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

*start\_item* [TreeItem](#) where the search is to begin, the search will not include *item*. If *item* is NULL, the search will begin from the first item in the list.

**Returns:**

Pointer to the first [TreeItem](#) in the list after *item* that has text matching *text*. If no item matches the criteria NULL is returned.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.306.2.9 TreeItem \* CEGUI::Tree::findFirstItemWithID (uint searchID)**

Search the list for an item with the specified text.

**Parameters:**

*text* [String](#) object containing the text to be searched for.

*start\_item* [TreeItem](#) where the search is to begin, the search will not include *item*. If *item* is NULL, the search will begin from the first item in the list.

**Returns:**

Pointer to the first [TreeItem](#) in the list after *item* that has text matching *text*. If no item matches the criteria NULL is returned.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.306.2.10 bool CEGUI::Tree::isTreeItemInList (const TreeItem \* item) const**

Return whether the specified [TreeItem](#) is in the List.

**Returns:**

true if [TreeItem](#) *item* is in the list, false if [TreeItem](#) *item* is not in the list.

**6.306.2.11 bool CEGUI::Tree::isVertScrollbarAlwaysShown (void) const**

Return whether the vertical scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.

**6.306.2.12 bool CEGUI::Tree::isHorzScrollbarAlwaysShown (void) const**

Return whether the horizontal scroll bar is always shown.

**Returns:**

- true if the scroll bar will always be shown even if it is not required.
- false if the scroll bar will only be shown when it is required.



**6.306.2.13 void CEGUI::Tree::initialise (void) [virtual]**

Initialise the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Factory](#) for each [Window](#) type.

**Returns:**

Nothing

**6.306.2.14 void CEGUI::Tree::resetList (void)**

Remove all items from the list.

Note that this will cause 'AutoDelete' items to be deleted.

**6.306.2.15 void CEGUI::Tree::addItem (TreeItem \* *item*)**

Add the given [TreeItem](#) to the list.

**Parameters:**

*item* Pointer to the [TreeItem](#) to be added to the list. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

**Returns:**

Nothing.

**6.306.2.16 void CEGUI::Tree::insertItem (TreeItem \* *item*, const TreeItem \* *position*)**

Insert an item into the list box after a specified item already in the list.

Note that if the list is sorted, the item may not end up in the requested position.

**Parameters:**

*item* Pointer to the [TreeItem](#) to be inserted. Note that it is the passed object that is added to the list, a copy is not made. If this parameter is NULL, nothing happens.

*position* Pointer to a [TreeItem](#) that *item* is to be inserted after. If this parameter is NULL, the item is inserted at the start of the list.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if no [TreeItem](#) *position* is attached to this list box.

**6.306.2.17 void CEGUI::Tree::removeItem (const TreeItem \* *item*)**

Removes the given item from the list box. If the item is has the auto delete state set, the item will be deleted.

**Parameters:**

*item* Pointer to the [TreeItem](#) that is to be removed. If *item* is not attached to this list box then nothing will happen.

**Returns:**

Nothing.

**6.306.2.18 void CEGUI::Tree::clearAllSelections (void)**

Clear the selected state for all items.

**Returns:**

Nothing.

**6.306.2.19 void CEGUI::Tree::setSortingEnabled (bool *setting*)**

Set whether the list should be sorted.

**Parameters:**

*setting* true if the list should be sorted, false if the list should not be sorted.

**Returns:**

Nothing.

**6.306.2.20 void CEGUI::Tree::setMultiselectEnabled (bool *setting*)**

Set whether the list should allow multiple selections or just a single selection.

**Parameters:**

*setting* true if the widget should allow multiple items to be selected, false if the widget should only allow a single selection.

**Returns:**

Nothing.

**6.306.2.21 void CEGUI::Tree::setShowVertScrollbar (bool *setting*)**

Set whether the vertical scroll bar should always be shown.

**Parameters:**

*setting* true if the vertical scroll bar should be shown even when it is not required. false if the vertical scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.306.2.22 void CEGUI::Tree::setShowHorzScrollbar (bool *setting*)**

Set whether the horizontal scroll bar should always be shown.

**Parameters:**

*setting* true if the horizontal scroll bar should be shown even when it is not required. false if the horizontal scroll bar should only be shown when it is required.

**Returns:**

Nothing.

**6.306.2.23 void CEGUI::Tree::setItemSelectState (TreeItem \* *item*, bool *state*)**

Set the select state of an attached [TreeItem](#).

This is the recommended way of selecting and deselecting items attached to a list box as it respects the multi-select mode setting. It is possible to modify the setting on TreeItems directly, but that approach does not respect the settings of the list box.

**Parameters:**

*item* The [TreeItem](#) to be affected. This item must be attached to the list box.  
*state* true to select the item, false to de-select the item.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.306.2.24 void CEGUI::Tree::setItemSelectState (size\_t *item\_index*, bool *state*)**

Set the select state of an attached [TreeItem](#).

This is the recommended way of selecting and deselecting items attached to a list box as it respects the multi-select mode setting. It is possible to modify the setting on TreeItems directly, but that approach does not respect the settings of the list box.

**Parameters:**

*item\_index* The zero based index of the [TreeItem](#) to be affected. This must be a valid index ( $0 \leq \text{index} < \text{getItemCount}()$ )

*state* true to select the item, false to de-select the item.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *item\_index* is out of range for the list box

**6.306.2.25 void CEGUI::Tree::setLookAndFeel (const String & look) [virtual]**

Set the LookAndFeel that should be used for this window.

**Parameters:**

*look* [String](#) object holding the name of the look to be assigned to the window.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if the look's feel specified by *look* does not exist.

**Note:**

Once a look's feel has been assigned it is locked - as it cannot be changed.

Reimplemented from [CEGUI::Window](#).

**6.306.2.26 void CEGUI::Tree::handleUpdatedItemData (void)**

Causes the list box to update its internal state after changes have been made to one or more attached [TreeItem](#) objects.

Client code must call this whenever it has made any changes to [TreeItem](#) objects already attached to the list box. If you are just adding items, or removed items to update them prior to re-adding them, there is no need to call this method.

**Returns:**

Nothing.

**6.306.2.27 void CEGUI::Tree::ensureItemIsVisible (const TreeItem \* item)**

Ensure the item at the specified index is visible within the list box.

**Parameters:**

*item* Pointer to the [TreeItem](#) to be made visible in the list box.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) thrown if *item* is not attached to this list box.

**6.306.2.28** `virtual Rect CEGUI::Tree::getTreeRenderArea (void) const` [inline, protected, virtual]

Return a [Rect](#) object describing, in un-clipped pixels, the window relative area that is to be used for rendering list items.

**Returns:**

[Rect](#) object describing the area of the [Window](#) to be used for rendering list box items.

**6.306.2.29** `virtual Scrollbar* CEGUI::Tree::createVertScrollbar (const String & name) const` [inline, protected, virtual]

create and return a pointer to a [Scrollbar](#) widget for use as vertical scroll bar

**Parameters:**

*name* [String](#) holding the name to be given to the created widget component.

**Returns:**

Pointer to a [Scrollbar](#) to be used for scrolling the list vertically.

**6.306.2.30** `virtual Scrollbar* CEGUI::Tree::createHorzScrollbar (const String & name) const` [inline, protected, virtual]

create and return a pointer to a [Scrollbar](#) widget for use as horizontal scroll bar

**Parameters:**

*name* [String](#) holding the name to be given to the created widget component.

**Returns:**

Pointer to a [Scrollbar](#) to be used for scrolling the list horizontally.

**6.306.2.31** `virtual void CEGUI::Tree::cacheTreeBaseImagery ()` [inline, protected, virtual]

Perform caching of the widget control frame and other 'static' areas. This method should not render the actual items. Note that the items are typically rendered to layer 3, other layers can be used for rendering imagery behind and infront of the items.

**Returns:**

Nothing.

**6.306.2.32** `bool CEGUI::Tree::getHeightToItemInList (const LBItemList & itemList, const TreeItem * treeItem, int itemDepth, float * height) const` [protected]

Clear the selected state for all items (implementation).

**Returns:**

true if treeItem was found in the search, false if it was not.

**6.306.2.33** `bool CEGUI::Tree::clearAllSelections_impl (void)` [protected]

Clear the selected state for all items (implementation).

**Returns:**

true if some selections were cleared, false nothing was changed.

**6.306.2.34** `TreeItem * CEGUI::Tree::getItemAtPoint (const Point & pt) const` [protected]

Return the [TreeItem](#) under the given window local pixel co-ordinate.

**Returns:**

[TreeItem](#) that is under window pixel co-ordinate *pt*, or NULL if no item is under that position.

**6.306.2.35** `bool CEGUI::Tree::resetList_impl (void)` [protected]

Remove all items from the list.

**Note:**

Note that this will cause 'AutoDelete' items to be deleted.

**Returns:**

- true if the list contents were changed.
- false if the list contents were not changed (list already empty).

**6.306.2.36 virtual bool CEGUI::Tree::testClassName\_impl (const String & *class\_name*) const**  
[inline, protected, virtual]

Return whether this window was inherited from the given class name at some point in the inheritance heirarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

Reimplemented from [CEGUI::Window](#).

**6.306.2.37 void CEGUI::Tree::populateRenderCache ()** [protected, virtual]

Update the rendering cache.

Populates the Window's [RenderCache](#) with imagery to be sent to the renderer.

Reimplemented from [CEGUI::Window](#).

**6.306.2.38 void CEGUI::Tree::onSized (WindowEventArgs & *e*)** [protected, virtual]

Handler called when the window's size changes.

**Parameters:**

*e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented from [CEGUI::Window](#).

**6.306.2.39 void CEGUI::Tree::onMouseButtonDown (MouseEventArgs & *e*)** [protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.306.2.40 void CEGUI::Tree::onMouseWheel (MouseEventArgs & *e*)** [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented from [CEGUI::Window](#).

**6.306.2.41** `void CEGUI::Tree::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

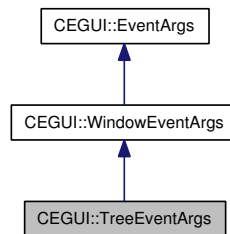
Reimplemented from [CEGUI::Window](#).



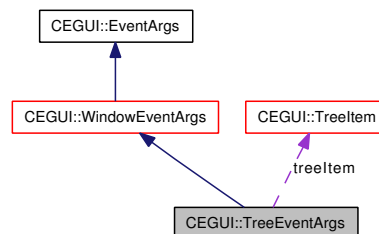
## 6.307 CEGUI::TreeEventArgs Class Reference

[EventArgs](#) based class that is used for objects passed to input event handlers concerning [Tree](#) events.

Inheritance diagram for CEGUI::TreeEventArgs:



Collaboration diagram for CEGUI::TreeEventArgs:



### Public Member Functions

- [TreeEventArgs](#) ([Window](#) \*wnd)

### Public Attributes

- [TreeItem](#) \* treeItem

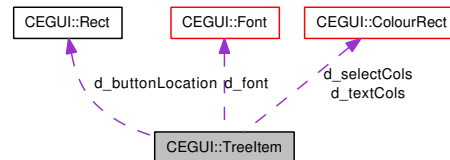
#### 6.307.1 Detailed Description

[EventArgs](#) based class that is used for objects passed to input event handlers concerning [Tree](#) events.

## 6.308 CEGUI::TreeItem Class Reference

Base class for list box items.

Collaboration diagram for CEGUI::TreeItem:



### Public Types

- typedef std::vector< [TreeItem](#) \* > **LBItemList**

### Public Member Functions

- [TreeItem](#) (const [String](#) &text, uint item\_id=0, void \*item\_data=0, bool disabled=false, bool auto\_delete=true)  
*base class constructor*
- virtual [~TreeItem](#) (void)  
*base class destructor*
- [Font](#) \* [getFont](#) (void) const  
*Return a pointer to the font being used by this [ListboxTextItem](#).*
- [ColourRect](#) [getTextColours](#) (void) const  
*Return the current colours used for text rendering.*
- void [setFont](#) ([Font](#) \*font)  
*Set the font to be used by this [ListboxTextItem](#).*
- void [setFont](#) (const [String](#) &font\_name)  
*Set the font to be used by this [ListboxTextItem](#).*
- void [setTextColours](#) (const [ColourRect](#) &cols)  
*Set the colours used for text rendering.*
- void [setTextColours](#) ([colour](#) top\_left\_colour, [colour](#) top\_right\_colour, [colour](#) bottom\_left\_colour, [colour](#) bottom\_right\_colour)  
*Set the colours used for text rendering.*
- void [setTextColours](#) ([colour](#) col)  
*Set the colours used for text rendering.*
- const [String](#) & [getText](#) (void) const  
*return the text string set for this list box item.*

- const [String](#) & **getTooltipText** (void) const
- uint **getID** (void) const  
*Return the current ID assigned to this list box item.*
- void \* **getUserData** (void) const  
*Return the pointer to any client assigned user data attached to this list box item.*
- bool **isSelected** (void) const  
*return whether this item is selected.*
- bool **isDisabled** (void) const  
*return whether this item is disabled.*
- bool **isAutoDeleted** (void) const  
*return whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.*
- const [Window](#) \* **getOwnerWindow** (void)  
*Get the owner window for this [TreeItem](#).*
- [ColourRect](#) **getSelectionColours** (void) const  
*Return the current colours used for selection highlighting.*
- const [Image](#) \* **getSelectionBrushImage** (void) const  
*Return the current selection highlighting brush.*
- void **setText** (const [String](#) &text)  
*set the text string for this list box item.*
- void **setTooltipText** (const [String](#) &text)
- void **setID** (uint item\_id)  
*Set the ID assigned to this list box item.*
- void **setUserData** (void \*item\_data)  
*Set the client assigned user data attached to this list box item.*
- void **setSelected** (bool setting)  
*set whether this item is selected.*
- void **setDisabled** (bool setting)  
*set whether this item is disabled.*
- void **setAutoDeleted** (bool setting)  
*Set whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.*
- void **setOwnerWindow** (const [Window](#) \*owner)  
*Set the owner window for this [TreeItem](#). This is called by all the list box widgets when an item is added or inserted.*

- void **setSelectionColours** (const **ColourRect** &cols)  
*Set the colours used for selection highlighting.*
- void **setSelectionColours** (**colour** top\_left\_colour, **colour** top\_right\_colour, **colour** bottom\_left\_colour, **colour** bottom\_right\_colour)  
*Set the colours used for selection highlighting.*
- void **setSelectionColours** (**colour** col)  
*Set the colours used for selection highlighting.*
- void **setSelectionBrushImage** (const **Image** \*image)  
*Set the selection highlighting brush image.*
- void **setSelectionBrushImage** (const **String** &imageset, const **String** &image)  
*Set the selection highlighting brush image.*
- void **setButtonLocation** (**Rect** &buttonOffset)  
*Tell the treeItem where its button is located. Calculated and set in Tree.cpp.*
- **Rect** & **getButtonLocation** (void)
- bool **getIsOpen** (void)
- void **toggleIsOpen** (void)
- **TreeItem** \* **getTreeItemFromIndex** (size\_t itemIndex)
- size\_t **getItemCount** (void) const
- **LBItemList** & **getItemList** (void)
- void **addItem** (**TreeItem** \*item)
- void **setIcon** (const **Image** &theIcon)
- virtual **Size** **getPixelSize** (void) const  
*Return the rendered pixel size of this list box item.*
- virtual void **draw** (const **Vector3** &position, float alpha, const **Rect** &clipper) const  
*Draw the list box item in its current state.*
- virtual void **draw** (**RenderCache** &cache, const **Rect** &targetRect, float zBase, float alpha, const **Rect** \*clipper) const
- virtual bool **operator<** (const **TreeItem** &rhs) const  
*Less-than operator; compares item texts.*
- virtual bool **operator>** (const **TreeItem** &rhs) const  
*Greater-than operator; compares item texts.*

## Static Public Attributes

- static const **colour** **DefaultTextColour** = 0xFFFFFFFF  
*Default text colour.*
- static const **colour** **DefaultSelectionColour** = 0xFF4444AA  
*Default selection brush colour.*

## Protected Member Functions

- [ColourRect](#) [getModulateAlphaColourRect](#) (const [ColourRect](#) &cols, float alpha) const  
*Return a [ColourRect](#) object describing the colours in cols after having their alpha component modulated by the value alpha.*
- [colour](#) [calculateModulatedAlphaColour](#) ([colour](#) col, float alpha) const  
*Return a [colour](#) value describing the [colour](#) specified by col after having its alpha component modulated by the value alpha.*

## Protected Attributes

- [String](#) [d\\_itemText](#)  
*Text for this list box item. If not rendered, this is still used for list sorting.*
- [String](#) [d\\_tooltipText](#)  
*Text for the individual tooltip of this item.*
- [uint](#) [d\\_itemID](#)  
*ID code assigned by client code. This has no meaning within the GUI system.*
- [void](#) \* [d\\_itemData](#)  
*Pointer to some client code data. This has no meaning within the GUI system.*
- [bool](#) [d\\_selected](#)  
*true if this item is selected. false if the item is not selected.*
- [bool](#) [d\\_disabled](#)  
*true if this item is disabled. false if the item is not disabled.*
- [bool](#) [d\\_autoDelete](#)  
*true if the system should destroy this item, false if client code will destroy the item.*
- [Rect](#) [d\\_buttonLocation](#)
- [const](#) [Window](#) \* [d\\_owner](#)  
*Pointer to the window that owns this item.*
- [ColourRect](#) [d\\_selectCols](#)  
*Colours used for selection highlighting.*
- [const](#) [Image](#) \* [d\\_selectBrush](#)  
*[Image](#) used for rendering selection.*
- [ColourRect](#) [d\\_textCols](#)  
*Colours used for rendering the text.*
- [Font](#) \* [d\\_font](#)  
*[Font](#) used for rendering text.*

- [Image \\* d\\_iconImage](#)  
*Icon to be displayed with this treeItem.*
- [LBItemList d\\_listItems](#)  
*list of items in this item's tree branch.*
- [bool d\\_isOpen](#)  
*true if the this item's tree branch is opened*

### 6.308.1 Detailed Description

Base class for list box items.

### 6.308.2 Member Function Documentation

#### 6.308.2.1 [Font \\* CEGUI::TreeItem::getFont \(void\) const](#)

Return a pointer to the font being used by this [ListboxTextItem](#).

This method will try a number of places to find a font to be used. If no font can be found, NULL is returned.

##### Returns:

[Font](#) to be used for rendering this item

#### 6.308.2.2 [ColourRect CEGUI::TreeItem::getTextColours \(void\) const](#) [inline]

Return the current colours used for text rendering.

##### Returns:

[ColourRect](#) object describing the currently set colours

#### 6.308.2.3 [void CEGUI::TreeItem::setFont \(Font \\*font\)](#) [inline]

Set the font to be used by this [ListboxTextItem](#).

##### Parameters:

*font* [Font](#) to be used for rendering this item

##### Returns:

Nothing

**6.308.2.4 void CEGUI::TreeItem::setFont (const String & *font\_name*)**

Set the font to be used by this [ListboxTextItem](#).

**Parameters:**

*font\_name* [String](#) object containing the name of the [Font](#) to be used for rendering this item

**Returns:**

Nothing

**6.308.2.5 void CEGUI::TreeItem::setTextColours (const ColourRect & *cols*)** `[inline]`

Set the colours used for text rendering.

**Parameters:**

*cols* [ColourRect](#) object describing the colours to be used.

**Returns:**

Nothing.

**6.308.2.6 void CEGUI::TreeItem::setTextColours (colour *top\_left\_colour*, colour *top\_right\_colour*, colour *bottom\_left\_colour*, colour *bottom\_right\_colour*)**

Set the colours used for text rendering.

**Parameters:**

*top\_left\_colour* Colour (as ARGB value) to be applied to the top-left corner of each text glyph rendered.

*top\_right\_colour* Colour (as ARGB value) to be applied to the top-right corner of each text glyph rendered.

*bottom\_left\_colour* Colour (as ARGB value) to be applied to the bottom-left corner of each text glyph rendered.

*bottom\_right\_colour* Colour (as ARGB value) to be applied to the bottom-right corner of each text glyph rendered.

**Returns:**

Nothing.

**6.308.2.7 void CEGUI::TreeItem::setTextColours (colour *col*)** `[inline]`

Set the colours used for text rendering.

**Parameters:**

*col* [colour](#) value to be used when rendering.

**Returns:**

Nothing.

**6.308.2.8   const String& CEGUI::TreeItem::getText (void) const   [inline]**

return the text string set for this list box item.

Note that even if the item does not render text, the text string can still be useful, since it is used for sorting list box items.

**Returns:**

[String](#) object containing the current text for the list box item.

**6.308.2.9   uint CEGUI::TreeItem::getID (void) const   [inline]**

Return the current ID assigned to this list box item.

Note that the system does not make use of this value, client code can assign any meaning it wishes to the ID.

**Returns:**

ID code currently assigned to this list box item

**6.308.2.10   void\* CEGUI::TreeItem::getUserData (void) const   [inline]**

Return the pointer to any client assigned user data attached to this list box item.

Note that the system does not make use of this data, client code can assign any meaning it wishes to the attached data.

**Returns:**

Pointer to the currently assigned user data.

**6.308.2.11   bool CEGUI::TreeItem::isSelected (void) const   [inline]**

return whether this item is selected.

**Returns:**

true if the item is selected, false if the item is not selected.

**6.308.2.12   bool CEGUI::TreeItem::isDisabled (void) const   [inline]**

return whether this item is disabled.

**Returns:**

true if the item is disabled, false if the item is enabled.



**6.308.2.13 bool CEGUI::TreeItem::isAutoDeleted (void) const** [inline]

return whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.

**Returns:**

true if the item object will be deleted by the system when the list box it is attached to is destroyed, or when the item is removed from the list. false if client code must destroy the item after it is removed from the list.

**6.308.2.14 const Window\* CEGUI::TreeItem::getOwnerWindow (void)** [inline]

Get the owner window for this [TreeItem](#).

The owner of a [TreeItem](#) is typically set by the list box widgets when an item is added or inserted.

**Returns:**

Ponter to the window that is considered the owner of this [TreeItem](#).

**6.308.2.15 ColourRect CEGUI::TreeItem::getSelectionColours (void) const** [inline]

Return the current colours used for selection highlighting.

**Returns:**

[ColourRect](#) object describing the currently set colours

**6.308.2.16 const Image\* CEGUI::TreeItem::getSelectionBrushImage (void) const** [inline]

Return the current selection highlighting brush.

**Returns:**

Pointer to the [Image](#) object currently used for selection highlighting.

**6.308.2.17 void CEGUI::TreeItem::setText (const String & text)** [inline]

set the text string for this list box item.

Note that even if the item does not render text, the text string can still be useful, since it is used for sorting list box items.

**Parameters:**

*text* [String](#) object containing the text to set for the list box item.

**Returns:**

Nothing.

**6.308.2.18 void CEGUI::TreeItem::setID (uint *item\_id*) [inline]**

Set the ID assigned to this list box item.

Note that the system does not make use of this value, client code can assign any meaning it wishes to the ID.

**Parameters:**

*item\_id* ID code to be assigned to this list box item

**Returns:**

Nothing.

**6.308.2.19 void CEGUI::TreeItem::setUserData (void \* *item\_data*) [inline]**

Set the client assigned user data attached to this list box item.

Note that the system does not make use of this data, client code can assign any meaning it wishes to the attached data.

**Parameters:**

*item\_data* Pointer to the user data to attach to this list item.

**Returns:**

Nothing.

**6.308.2.20 void CEGUI::TreeItem::setSelected (bool *setting*) [inline]**

set whether this item is selected.

**Parameters:**

*setting* true if the item is selected, false if the item is not selected.

**Returns:**

Nothing.

**6.308.2.21 void CEGUI::TreeItem::setDisabled (bool *setting*) [inline]**

set whether this item is disabled.

**Parameters:**

*setting* true if the item is disabled, false if the item is enabled.

**Returns:**

Nothing.

**6.308.2.22 void CEGUI::TreeItem::setAutoDeleted (bool *setting*)** [inline]

Set whether this item will be automatically deleted when the list box it is attached to is destroyed, or when the item is removed from the list box.

**Parameters:**

*setting* true if the item object should be deleted by the system when the list box it is attached to is destroyed, or when the item is removed from the list. false if client code will destroy the item after it is removed from the list.

**Returns:**

Nothing.

**6.308.2.23 void CEGUI::TreeItem::setOwnerWindow (const Window \* *owner*)** [inline]

Set the owner window for this [TreeItem](#). This is called by all the list box widgets when an item is added or inserted.

**Parameters:**

*owner* Ponter to the window that should be considered the owner of this [TreeItem](#).

**Returns:**

Nothing

**6.308.2.24 void CEGUI::TreeItem::setSelectionColours (const ColourRect & *cols*)** [inline]

Set the colours used for selection highlighting.

**Parameters:**

*cols* [ColourRect](#) object describing the colours to be used.

**Returns:**

Nothing.

**6.308.2.25 void CEGUI::TreeItem::setSelectionColours (colour *top\_left\_colour*, colour *top\_right\_colour*, colour *bottom\_left\_colour*, colour *bottom\_right\_colour*)**

Set the colours used for selection highlighting.

**Parameters:**

*top\_left\_colour* Colour (as ARGB value) to be applied to the top-left corner of the selection area.

*top\_right\_colour* Colour (as ARGB value) to be applied to the top-right corner of the selection area.

*bottom\_left\_colour* Colour (as ARGB value) to be applied to the bottom-left corner of the selection area.

*bottom\_right\_colour* Colour (as ARGB value) to be applied to the bottom-right corner of the selection area.

**Returns:**

Nothing.

**6.308.2.26 void CEGUI::TreeItem::setSelectionColours (colour *col*) [inline]**

Set the colours used for selection highlighting.

**Parameters:**

*col* *colour* value to be used when rendering.

**Returns:**

Nothing.

**6.308.2.27 void CEGUI::TreeItem::setSelectionBrushImage (const Image \* *image*) [inline]**

Set the selection highlighting brush image.

**Parameters:**

*image* Pointer to the [Image](#) object to be used for selection highlighting.

**Returns:**

Nothing.

**6.308.2.28 void CEGUI::TreeItem::setSelectionBrushImage (const String & *imageset*, const String & *image*)**

Set the selection highlighting brush image.

**Parameters:**

*imageset* Name of the imageset containing the image to be used.

*image* Name of the image to be used

**Returns:**

Nothing.

**6.308.2.29 void CEGUI::TreeItem::setButtonLocation (Rect & *buttonOffset*) [inline]**

Tell the treeItem where its button is located. Calculated and set in Tree.cpp.

**Parameters:**

*buttonOffset* Location of the button in screenspace.

**6.308.2.30** `Size CEGUI::TreeItem::getPixelSize (void) const` [virtual]

Return the rendered pixel size of this list box item.

**Returns:**

[Size](#) object describing the size of the list box item in pixels.

**6.308.2.31** `void CEGUI::TreeItem::draw (const Vector3 & position, float alpha, const Rect & clipper) const` [virtual]

Draw the list box item in its current state.

**Parameters:**

*position* Vecor3 object describing the upper-left corner of area that should be rendered in to for the draw operation.

*alpha* Alpha value to be used when rendering the item (between 0.0f and 1.0f).

*clipper* [Rect](#) object describing the clipping rectangle for the draw operation.

**Returns:**

Nothing.

## 6.309 CEGUI::UDim Class Reference

Class representing a unified dimension; that is a dimension that has both a relative 'scale' portion and and absolute 'offset' portion.

### Public Member Functions

- **UDim** (float scale, float offset)
- float **asAbsolute** (float base) const
- float **asRelative** (float base) const
- **UDim operator+** (const **UDim** &other) const
- **UDim operator-** (const **UDim** &other) const
- **UDim operator/** (const **UDim** &other) const
- **UDim operator \*** (const **UDim** &other) const
- const **UDim** & **operator+=** (const **UDim** &other)
- const **UDim** & **operator-=** (const **UDim** &other)
- const **UDim** & **operator/=** (const **UDim** &other)
- const **UDim** & **operator \*=** (const **UDim** &other)
- bool **operator==** (const **UDim** &other) const
- bool **operator!=** (const **UDim** &other) const

### Public Attributes

- float **d\_scale**
- float **d\_offset**

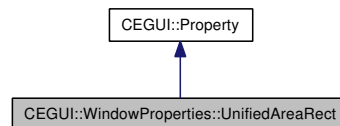
### 6.309.1 Detailed Description

Class representing a unified dimension; that is a dimension that has both a relative 'scale' portion and and absolute 'offset' portion.

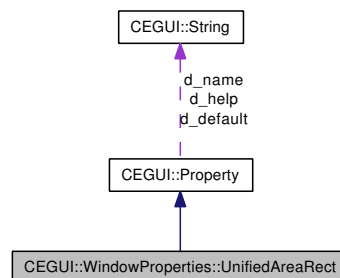
## 6.310 CEGUI::WindowProperties::UnifiedAreaRect Class Reference

[Property](#) to access the unified area rectangle of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedAreaRect:



Collaboration diagram for CEGUI::WindowProperties::UnifiedAreaRect:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.310.1 Detailed Description

[Property](#) to access the unified area rectangle of the window.

#### Usage:

- Name: [UnifiedAreaRect](#)
- Format: "{ { [ls],[lo] }, { [ts],[to] }, { [rs],[ro] }, { [bs],[bo] } }"

#### Where:

- [ls] is a floating point value describing the relative scale value for the left edge.
- [lo] is a floating point value describing the absolute offset value for the left edge.
- [ts] is a floating point value describing the relative scale value for the top edge.
- [to] is a floating point value describing the absolute offset value for the top edge.

- [rs] is a floating point value describing the relative scale value for the right edge.
- [ro] is a floating point value describing the absolute offset value for the right edge.
- [bs] is a floating point value describing the relative scale value for the bottom edge.
- [bo] is a floating point value describing the absolute offset value for the bottom edge.

## 6.310.2 Member Function Documentation

### 6.310.2.1 `String CEGUI::WindowProperties::UnifiedAreaRect::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.310.2.2 `void CEGUI::WindowProperties::UnifiedAreaRect::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

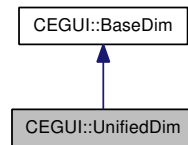
Implements [CEGUI::Property](#).



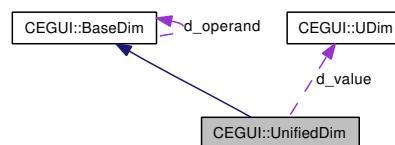
## 6.311 CEGUI::UnifiedDim Class Reference

**Dimension** type that represents an Unified dimension. Implements **BaseDim** interface.

Inheritance diagram for CEGUI::UnifiedDim:



Collaboration diagram for CEGUI::UnifiedDim:



### Public Member Functions

- **UnifiedDim** (const **UDim** &value, **DimensionType** dim)  
*Constructor.*

### Protected Member Functions

- float **getValue\_impl** (const **Window** &wnd) const  
*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically.*
- float **getValue\_impl** (const **Window** &wnd, const **Rect** &container) const  
*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically by **BaseDim**.*
- void **writeXMLElementName\_impl** (**XMLSerializer** &xml\_stream) const  
*Implementataion method to output real xml element name.*
- void **writeXMLElementAttributes\_impl** (**XMLSerializer** &xml\_stream) const  
*Implementataion method to create the element attributes.*
- **BaseDim** \* **clone\_impl** () const  
*Implementataion method to return a clone of this sub-class of **BaseDim**. This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by **BaseDim**.*

#### 6.311.1 Detailed Description

**Dimension** type that represents an Unified dimension. Implements **BaseDim** interface.

## 6.311.2 Constructor & Destructor Documentation

### 6.311.2.1 CEGUI::UnifiedDim::UnifiedDim (const UDim & *value*, DimensionType *dim*)

Constructor.

**Parameters:**

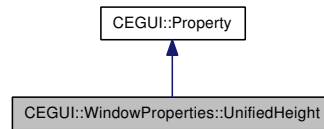
*value* [UDim](#) holding the value to assign to this [UnifiedDim](#).

*dim* DimensionType value indicating what this [UnifiedDim](#) is to represent. This is required because we need to know what part of the base [Window](#) that the [UDim](#) scale component is to operate against.

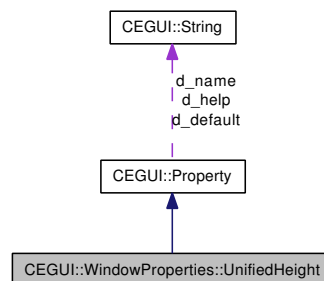
## 6.312 CEGUI::WindowProperties::UnifiedHeight Class Reference

[Property](#) to access the unified height of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedHeight:



Collaboration diagram for CEGUI::WindowProperties::UnifiedHeight:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.312.1 Detailed Description

[Property](#) to access the unified height of the window.

**Usage:**

- Name: [UnifiedHeight](#)
- Format: "{[s],[o]}"

**Where:**

- [s] is a floating point value describing the relative scale value for the height.
- [o] is a floating point value describing the absolute offset value for the height.

## 6.312.2 Member Function Documentation

### 6.312.2.1 `String CEGUI::WindowProperties::UnifiedHeight::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.312.2.2 `void CEGUI::WindowProperties::UnifiedHeight::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

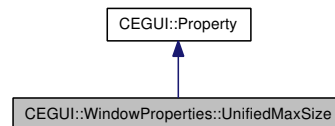
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

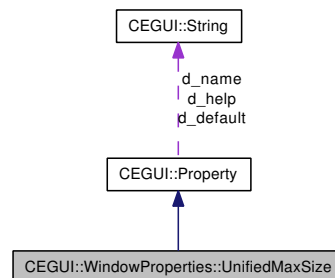
## 6.313 CEGUI::WindowProperties::UnifiedMaxSize Class Reference

[Property](#) to access the unified maximum size of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedMaxSize:



Collaboration diagram for CEGUI::WindowProperties::UnifiedMaxSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.313.1 Detailed Description

[Property](#) to access the unified maximum size of the window.

#### Usage:

- Name: [UnifiedMaxSize](#)
- Format: "{ { [ws],[wo] }, { [hs],[ho] } }"

#### Where:

- [ws] is a floating point value describing the relative scale value for the maximum width.
- [wo] is a floating point value describing the absolute offset value for the maximum width.
- [hs] is a floating point value describing the relative scale value for the maximum height.
- [ho] is a floating point value describing the absolute offset value for the maximum height.

## 6.313.2 Member Function Documentation

### 6.313.2.1 `String CEGUI::WindowProperties::UnifiedMaxSize::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.313.2.2 `void CEGUI::WindowProperties::UnifiedMaxSize::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

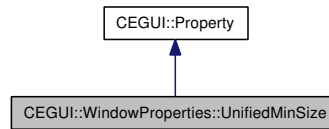
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

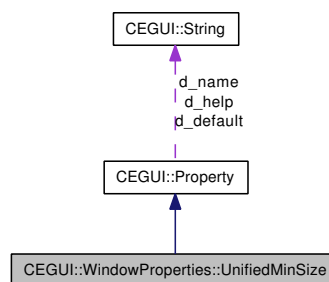
## 6.314 CEGUI::WindowProperties::UnifiedMinSize Class Reference

[Property](#) to access the unified minimum size of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedMinSize:



Collaboration diagram for CEGUI::WindowProperties::UnifiedMinSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.314.1 Detailed Description

[Property](#) to access the unified minimum size of the window.

Usage:

- Name: [UnifiedMinSize](#)
- Format: "{ { [ws],[wo] }, { [hs],[ho] } }"

Where:

- [ws] is a floating point value describing the relative scale value for the minimum width.
- [wo] is a floating point value describing the absolute offset value for the minimum width.
- [hs] is a floating point value describing the relative scale value for the minimum height.
- [ho] is a floating point value describing the absolute offset value for the minimum height.

## 6.314.2 Member Function Documentation

### 6.314.2.1 `String CEGUI::WindowProperties::UnifiedMinSize::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.314.2.2 `void CEGUI::WindowProperties::UnifiedMinSize::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

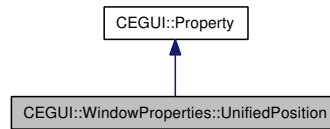
Implements [CEGUI::Property](#).



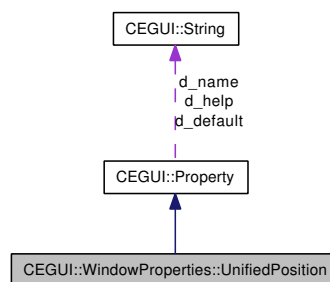
## 6.315 CEGUI::WindowProperties::UnifiedPosition Class Reference

[Property](#) to access the unified position of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedPosition:



Collaboration diagram for CEGUI::WindowProperties::UnifiedPosition:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.315.1 Detailed Description

[Property](#) to access the unified position of the window.

Usage:

- Name: [UnifiedPosition](#)
- Format: "[{[xs],[xo]},{[ys],[yo]}]"

Where:

- [xs] is a floating point value describing the relative scale value for the position x-coordinate.
- [xo] is a floating point value describing the absolute offset value for the position x-coordinate.
- [ys] is a floating point value describing the relative scale value for the position y-coordinate.
- [yo] is a floating point value describing the absolute offset value for the position y-coordinate.

## 6.315.2 Member Function Documentation

### 6.315.2.1 `String CEGUI::WindowProperties::UnifiedPosition::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.315.2.2 `void CEGUI::WindowProperties::UnifiedPosition::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

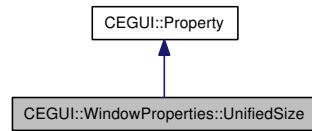
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

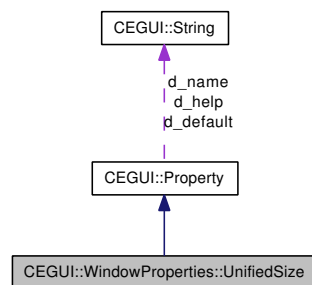
## 6.316 CEGUI::WindowProperties::UnifiedSize Class Reference

[Property](#) to access the unified position of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedSize:



Collaboration diagram for CEGUI::WindowProperties::UnifiedSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.316.1 Detailed Description

[Property](#) to access the unified position of the window.

**Usage:**

- Name: [UnifiedSize](#)
- Format: "{ { [ws],[wo] }, { [hs],[ho] } }"

**Where:**

- [ws] is a floating point value describing the relative scale value for the width.
- [wo] is a floating point value describing the absolute offset value for the width.
- [hs] is a floating point value describing the relative scale value for the height.
- [ho] is a floating point value describing the absolute offset value for the height.

## 6.316.2 Member Function Documentation

### 6.316.2.1 **String CEGUI::WindowProperties::UnifiedSize::get (const PropertyReceiver \* *receiver*) const** [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.316.2.2 **void CEGUI::WindowProperties::UnifiedSize::set (PropertyReceiver \* *receiver*, const String & *value*)** [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

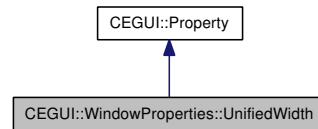
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

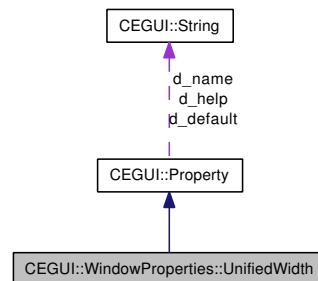
## 6.317 CEGUI::WindowProperties::UnifiedWidth Class Reference

[Property](#) to access the unified width of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedWidth:



Collaboration diagram for CEGUI::WindowProperties::UnifiedWidth:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.317.1 Detailed Description

[Property](#) to access the unified width of the window.

#### Usage:

- Name: [UnifiedWidth](#)
- Format: "{[s],[o]}"

#### Where:

- [s] is a floating point value describing the relative scale value for the width.
- [o] is a floating point value describing the absolute offset value for the width.

## 6.317.2 Member Function Documentation

### 6.317.2.1 `String CEGUI::WindowProperties::UnifiedWidth::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.317.2.2 `void CEGUI::WindowProperties::UnifiedWidth::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

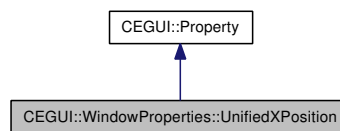
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

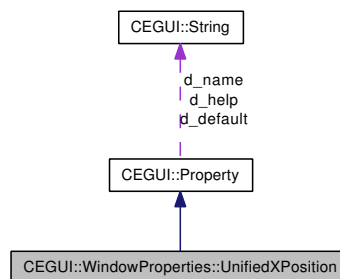
## 6.318 CEGUI::WindowProperties::UnifiedXPosition Class Reference

[Property](#) to access the unified position x-coordinate of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedXPosition:



Collaboration diagram for CEGUI::WindowProperties::UnifiedXPosition:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

### 6.318.1 Detailed Description

[Property](#) to access the unified position x-coordinate of the window.

Usage:

- Name: [UnifiedXPosition](#)
- Format: "[s],[o]"

Where:

- [s] is a floating point value describing the relative scale value for the position x-coordinate.
- [o] is a floating point value describing the absolute offset value for the position x-coordinate.

## 6.318.2 Member Function Documentation

### 6.318.2.1 `String CEGUI::WindowProperties::UnifiedXPosition::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.318.2.2 `void CEGUI::WindowProperties::UnifiedXPosition::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

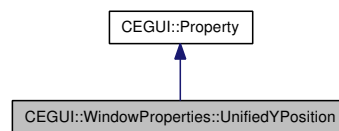
Implements [CEGUI::Property](#).



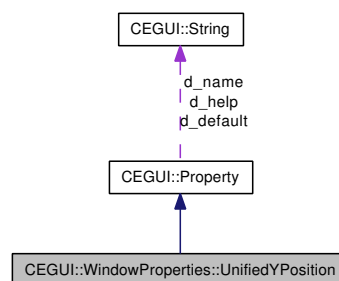
## 6.319 CEGUI::WindowProperties::UnifiedYPosition Class Reference

[Property](#) to access the unified position y-coordinate of the window.

Inheritance diagram for CEGUI::WindowProperties::UnifiedYPosition:



Collaboration diagram for CEGUI::WindowProperties::UnifiedYPosition:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
Return the current value of the [Property](#) as a [String](#).
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
Sets the value of the property.

### 6.319.1 Detailed Description

[Property](#) to access the unified position y-coordinate of the window.

Usage:

- Name: [UnifiedYPosition](#)
- Format: "[s],[o]"

Where:

- [s] is a floating point value describing the relative scale value for the position y-coordinate.
- [o] is a floating point value describing the absolute offset value for the position y-coordinate.

## 6.319.2 Member Function Documentation

### 6.319.2.1 `String CEGUI::WindowProperties::UnifiedYPosition::get (const PropertyReceiver * receiver) const` [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.319.2.2 `void CEGUI::WindowProperties::UnifiedYPosition::set (PropertyReceiver * receiver, const String & value)` [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

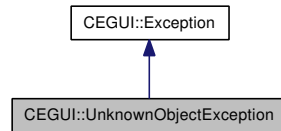
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

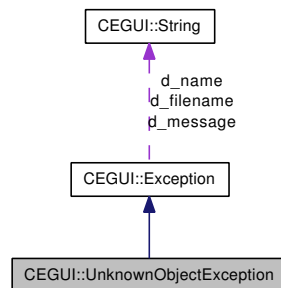
## 6.320 CEGUI::UnknownObjectException Class Reference

[Exception](#) class used when a request was made using a name of an unknown object.

Inheritance diagram for CEGUI::UnknownObjectException:



Collaboration diagram for CEGUI::UnknownObjectException:



### Public Member Functions

- [UnknownObjectException](#) (const [String](#) &message, const [String](#) &file="unknown", int line=0)  
*Constructor that is responsible for logging the unknown object exception by calling the base class.*

### 6.320.1 Detailed Description

[Exception](#) class used when a request was made using a name of an unknown object.

### 6.320.2 Constructor & Destructor Documentation

#### 6.320.2.1 CEGUI::UnknownObjectException::UnknownObjectException (const [String](#) &message, const [String](#) &file = "unknown", int line = 0) [inline]

Constructor that is responsible for logging the unknown object exception by calling the base class.

#### Parameters:

- message* [String](#) object describing the reason for the unknown object exception being thrown.
- filename* [String](#) object containing the name of the file where the unknown object exception occurred.
- line* Integer representing the line number where the unknown object exception occurred.

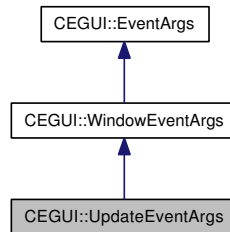
**Remarks:**

The unknown object exception name is automatically passed to the base class as "CEGUI::UnknownObjectException".

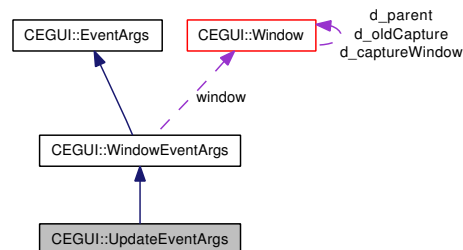
## 6.321 CEGUI::UpdateEventArgs Class Reference

[WindowEventArgs](#) class that is primarily used by lua scripts.

Inheritance diagram for CEGUI::UpdateEventArgs:



Collaboration diagram for CEGUI::UpdateEventArgs:



### Public Member Functions

- **UpdateEventArgs** ([Window](#) \*[window](#), float tsIf)

### Public Attributes

- float [d\\_timeSinceLastFrame](#)  
*Time since the last frame update.*

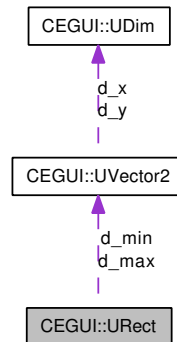
#### 6.321.1 Detailed Description

[WindowEventArgs](#) class that is primarily used by lua scripts.

## 6.322 CEGUI::URect Class Reference

Area rectangle class built using unified dimensions (UDims).

Collaboration diagram for CEGUI::URect:



### Public Member Functions

- **URect** (const [UVector2](#) &min, const [UVector2](#) &max)
- **URect** (const [UDim](#) &left, const [UDim](#) &top, const [UDim](#) &right, const [UDim](#) &bottom)
- **Rect asAbsolute** (const [Size](#) &base) const
- **Rect asRelative** (const [Size](#) &base) const
- const [UVector2](#) &**getPosition** () const
- [UVector2](#) **getSize** () const
- [UDim](#) **getWidth** () const
- [UDim](#) **getHeight** () const
- void **setPosition** (const [UVector2](#) &pos)
- void **setSize** (const [UVector2](#) &sz)
- void **setWidth** (const [UDim](#) &w)
- void **setHeight** (const [UDim](#) &h)
- void **offset** (const [UVector2](#) &sz)

### Public Attributes

- [UVector2](#) **d\_min**
- [UVector2](#) **d\_max**

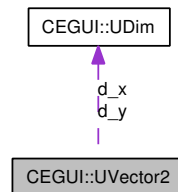
#### 6.322.1 Detailed Description

Area rectangle class built using unified dimensions (UDims).

## 6.323 CEGUI::UVector2 Class Reference

Two dimensional vector class built using unified dimensions (UDims). The [UVector2](#) class is used for representing both positions and sizes.

Collaboration diagram for CEGUI::UVector2:



### Public Member Functions

- **UVector2** (const [UDim](#) &x, const [UDim](#) &y)
- **Vector2 asAbsolute** (const [Size](#) &base) const
- **Vector2 asRelative** (const [Size](#) &base) const
- **UVector2 operator+** (const [UVector2](#) &other) const
- **UVector2 operator-** (const [UVector2](#) &other) const
- **UVector2 operator/** (const [UVector2](#) &other) const
- **UVector2 operator \*** (const [UVector2](#) &other) const
- const [UVector2](#) & **operator+=** (const [UVector2](#) &other)
- const [UVector2](#) & **operator-=** (const [UVector2](#) &other)
- const [UVector2](#) & **operator/=** (const [UVector2](#) &other)
- const [UVector2](#) & **operator \*=** (const [UVector2](#) &other)
- bool **operator==** (const [UVector2](#) &other) const
- bool **operator!=** (const [UVector2](#) &other) const

### Public Attributes

- [UDim](#) `d_x`
- [UDim](#) `d_y`

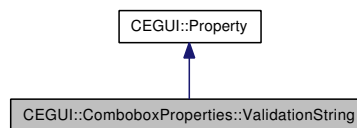
#### 6.323.1 Detailed Description

Two dimensional vector class built using unified dimensions (UDims). The [UVector2](#) class is used for representing both positions and sizes.

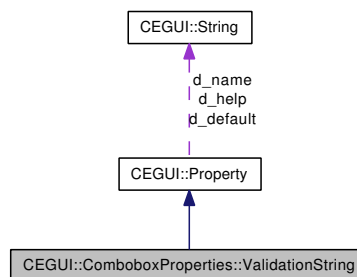
## 6.324 CEGUI::ComboboxProperties::ValidationString Class Reference

[Property](#) to access the string used for regular expression validation of the edit box text.

Inheritance diagram for CEGUI::ComboboxProperties::ValidationString:



Collaboration diagram for CEGUI::ComboboxProperties::ValidationString:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.324.1 Detailed Description

[Property](#) to access the string used for regular expression validation of the edit box text.

##### Usage:

- Name: [ValidationString](#)
- Format: "[text]"

##### Where:

- [Text] is the string used for validating text entry.



## 6.324.2 Member Function Documentation

### 6.324.2.1 String CEGUI::ComboboxProperties::ValidationString::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.324.2.2 void CEGUI::ComboboxProperties::ValidationString::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

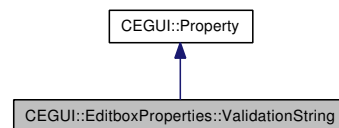
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

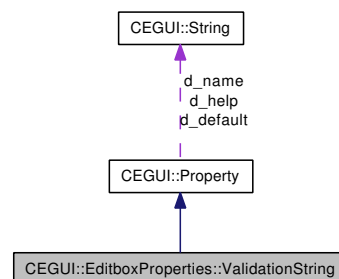
## 6.325 CEGUI::EditboxProperties::ValidationString Class Reference

[Property](#) to access the string used for regular expression validation of the edit box text.

Inheritance diagram for CEGUI::EditboxProperties::ValidationString:



Collaboration diagram for CEGUI::EditboxProperties::ValidationString:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.325.1 Detailed Description

[Property](#) to access the string used for regular expression validation of the edit box text.

#### Usage:

- Name: [ValidationString](#)
- Format: "[text]"

#### Where:

- [Text] is the string used for validating text entry.

## 6.325.2 Member Function Documentation

### 6.325.2.1 String CEGUI::EditboxProperties::ValidationString::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.325.2.2 void CEGUI::EditboxProperties::ValidationString::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.326 CEGUI::Vector2 Class Reference

Class used as a two dimensional vector (aka a Point).

### Public Member Functions

- **Vector2** (float x, float y)
- **Vector2** & **operator** \*= (const **Vector2** &vec)
- **Vector2** & **operator** /= (const **Vector2** &vec)
- **Vector2** & **operator** += (const **Vector2** &vec)
- **Vector2** & **operator** -= (const **Vector2** &vec)
- **Vector2** **operator** + (const **Vector2** &vec) const
- **Vector2** **operator** - (const **Vector2** &vec) const
- **Vector2** **operator** \* (const **Vector2** &vec) const
- bool **operator** == (const **Vector2** &vec) const
- bool **operator** != (const **Vector2** &vec) const
- **Size** **asSize** () const

### Public Attributes

- float **d\_x**
- float **d\_y**

#### 6.326.1 Detailed Description

Class used as a two dimensional vector (aka a Point).

## 6.327 CEGUI::Vector3 Class Reference

Class used as a three dimensional vector.

### Public Member Functions

- **Vector3** (float x, float y, float z)

### Public Attributes

- float **d\_x**
- float **d\_y**
- float **d\_z**

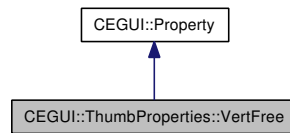
### 6.327.1 Detailed Description

Class used as a three dimensional vector.

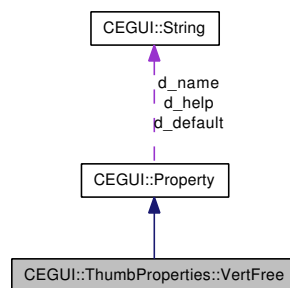
## 6.328 CEGUI::ThumbProperties::VertFree Class Reference

[Property](#) to access the state the setting to free the thumb vertically.

Inheritance diagram for CEGUI::ThumbProperties::VertFree:



Collaboration diagram for CEGUI::ThumbProperties::VertFree:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.328.1 Detailed Description

[Property](#) to access the state the setting to free the thumb vertically.

Usage:

- Name: [VertFree](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the thumb will be free (movable) vertically.
- "False" to indicate the thumb will be fixed vertically.

## 6.328.2 Member Function Documentation

### 6.328.2.1 String CEGUI::ThumbProperties::VertFree::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.328.2.2 void CEGUI::ThumbProperties::VertFree::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

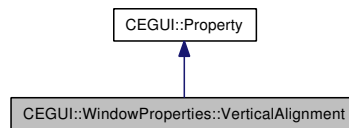
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

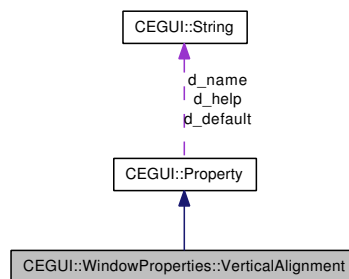
## 6.329 CEGUI::WindowProperties::VerticalAlignment Class Reference

[Property](#) to access the vertical alignment setting for the window.

Inheritance diagram for CEGUI::WindowProperties::VerticalAlignment:



Collaboration diagram for CEGUI::WindowProperties::VerticalAlignment:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.329.1 Detailed Description

[Property](#) to access the vertical alignment setting for the window.

Usage:

- Name: [VerticalAlignment](#)
- Format: "[text]".

Where [Text] is:

- "Top" to indicate the windows position is an offset of its top edge from its parents top edge.
- "Centre" to indicate the windows position is an offset of its centre point from its parents centre point.
- "Bottom" to indicate the windows position is an offset of its bottom edge from its parents bottom edge.



## 6.329.2 Member Function Documentation

### 6.329.2.1 String CEGUI::WindowProperties::VerticalAlignment::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.329.2.2 void CEGUI::WindowProperties::VerticalAlignment::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

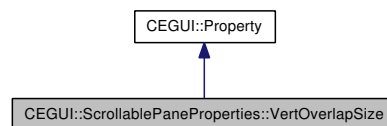
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

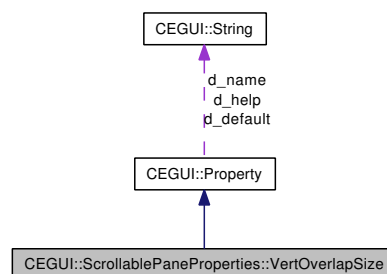
## 6.330 CEGUI::ScrollablePaneProperties::VertOverlapSize Class Reference

[Property](#) to access the overlap size for the vertical [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollablePaneProperties::VertOverlapSize:



Collaboration diagram for CEGUI::ScrollablePaneProperties::VertOverlapSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.330.1 Detailed Description

[Property](#) to access the overlap size for the vertical [Scrollbar](#).

#### Usage:

- Name: [VertOverlapSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the per-page overlap (as a fraction of one page).

## 6.330.2 Member Function Documentation

### 6.330.2.1 String CEGUI::ScrollablePaneProperties::VertOverlapSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.330.2.2 void CEGUI::ScrollablePaneProperties::VertOverlapSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

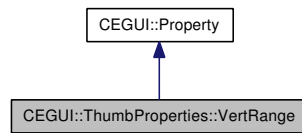
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

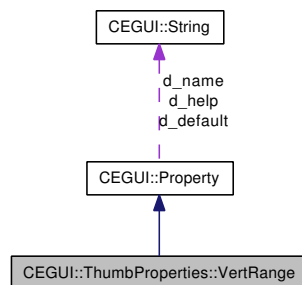
## 6.331 CEGUI::ThumbProperties::VertRange Class Reference

[Property](#) to access the vertical movement range for the thumb.

Inheritance diagram for CEGUI::ThumbProperties::VertRange:



Collaboration diagram for CEGUI::ThumbProperties::VertRange:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.331.1 Detailed Description

[Property](#) to access the vertical movement range for the thumb.

#### Usage:

- Name: [VertRange](#)
- Format: "min:[float] max:[float]".

#### Where:

- min:[float] specifies the minimum vertical setting (position) for the thumb, specified using the active metrics system for the window.
- max:[float] specifies the maximum vertical setting (position) for the thumb, specified using the active metrics system for the window.

## 6.331.2 Member Function Documentation

### 6.331.2.1 String CEGUI::ThumbProperties::VertRange::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.331.2.2 void CEGUI::ThumbProperties::VertRange::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

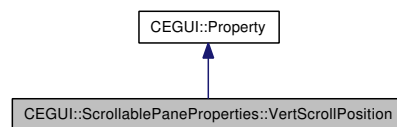
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

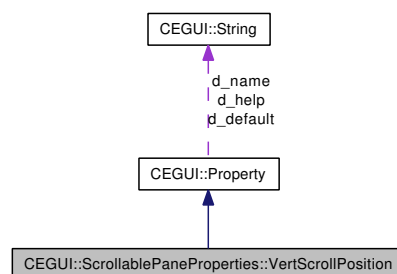
## 6.332 CEGUI::ScrollablePaneProperties::VertScrollPosition Class Reference

[Property](#) to access the scroll position of the vertical [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollablePaneProperties::VertScrollPosition:



Collaboration diagram for CEGUI::ScrollablePaneProperties::VertScrollPosition:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.332.1 Detailed Description

[Property](#) to access the scroll position of the vertical [Scrollbar](#).

#### Usage:

- Name: [VertScrollPosition](#)
- Format: "[float]".

#### Where:

- [float] specifies the current scroll position / value of the vertical [Scrollbar](#) (as a fraction of the whole).

## 6.332.2 Member Function Documentation

### 6.332.2.1 String CEGUI::ScrollablePaneProperties::VertScrollPosition::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.332.2.2 void CEGUI::ScrollablePaneProperties::VertScrollPosition::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

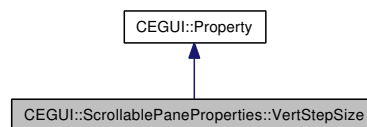
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

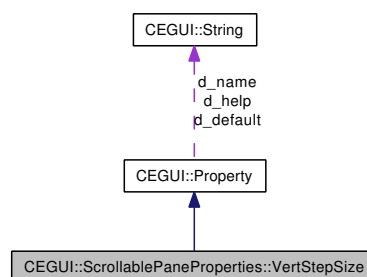
## 6.333 CEGUI::ScrollablePaneProperties::VertStepSize Class Reference

[Property](#) to access the step size for the vertical [Scrollbar](#).

Inheritance diagram for CEGUI::ScrollablePaneProperties::VertStepSize:



Collaboration diagram for CEGUI::ScrollablePaneProperties::VertStepSize:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.333.1 Detailed Description

[Property](#) to access the step size for the vertical [Scrollbar](#).

#### Usage:

- Name: [VertStepSize](#)
- Format: "[float]".

#### Where:

- [float] specifies the size of the increase/decrease button step for the vertical scrollbar (as a fraction of 1 page).



## 6.333.2 Member Function Documentation

### 6.333.2.1 String CEGUI::ScrollablePaneProperties::VertStepSize::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.333.2.2 void CEGUI::ScrollablePaneProperties::VertStepSize::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

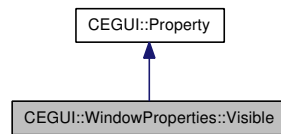
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

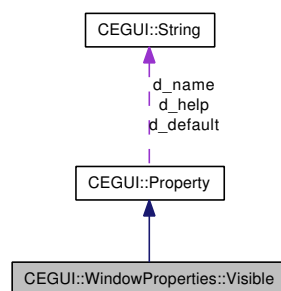
## 6.334 CEGUI::WindowProperties::Visible Class Reference

[Property](#) to access window [Visible](#) setting.

Inheritance diagram for CEGUI::WindowProperties::Visible:



Collaboration diagram for CEGUI::WindowProperties::Visible:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- bool [isDefault](#) (const [PropertyReceiver](#) \*receiver) const  
*Returns whether the property is at it's default value.*

### 6.334.1 Detailed Description

[Property](#) to access window [Visible](#) setting.

This property offers access to the visible setting for the window.

**Usage:**

- Name: [Visible](#)
- Format: "[text]".

**Where [Text] is:**

- "True" to indicate the [Window](#) is visible.
- "False" to indicate the [Window](#) is not visible.

## 6.334.2 Member Function Documentation

### 6.334.2.1 String CEGUI::WindowProperties::Visible::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.334.2.2 void CEGUI::WindowProperties::Visible::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

### 6.334.2.3 bool CEGUI::WindowProperties::Visible::isDefault (const PropertyReceiver \* *receiver*) const [virtual]

Returns whether the property is at it's default value.

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

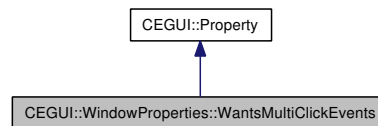
- true if the property has it's default value.
- false if the property has been modified from it's default value.

Reimplemented from [CEGUI::Property](#).

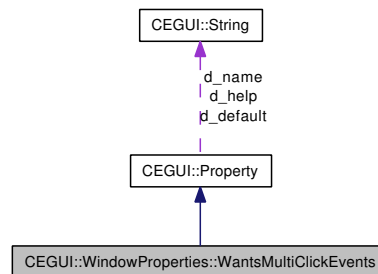
## 6.335 CEGUI::WindowProperties::WantsMultiClickEvents Class Reference

[Property](#) to control whether the window will receive double/triple-click events.

Inheritance diagram for CEGUI::WindowProperties::WantsMultiClickEvents:



Collaboration diagram for CEGUI::WindowProperties::WantsMultiClickEvents:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

### 6.335.1 Detailed Description

[Property](#) to control whether the window will receive double/triple-click events.

This property offers access to the setting that controls whether a window will receive double-click and triple-click events, or whether the window will receive multiple single mouse button down events instead.

#### Usage:

- Name: [WantsMultiClickEvents](#)
- Format: "[text]".

#### Where [Text] is:

- "True" to indicate the [Window](#) wants double-click and triple-click events.
- "False" to indicate the [Window](#) wants multiple single mouse button down events.

## 6.335.2 Member Function Documentation

### 6.335.2.1 String CEGUI::WindowProperties::WantsMultiClickEvents::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.335.2.2 void CEGUI::WindowProperties::WantsMultiClickEvents::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

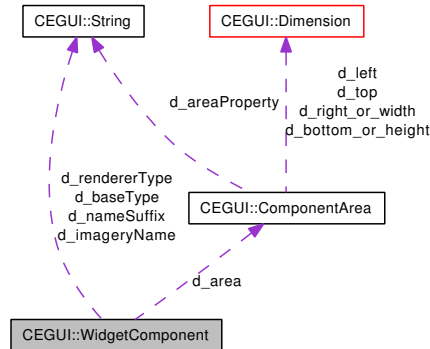
[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.336 CEGUI::WidgetComponent Class Reference

Class that encapsulates information regarding a sub-widget required for a widget.

Collaboration diagram for CEGUI::WidgetComponent:



### Public Member Functions

- **WidgetComponent** (const [String](#) &type, const [String](#) &look, const [String](#) &suffix, const [String](#) &renderer)
- void **create** ([Window](#) &parent) const

*Create an instance of this widget component adding it as a child to the given [Window](#).*

- const [ComponentArea](#) & **getComponentArea** () const
- void **setComponentArea** (const [ComponentArea](#) &area)
- const [String](#) & **getBaseWidgetType** () const
- void **setBaseWidgetType** (const [String](#) &type)
- const [String](#) & **getWidgetLookName** () const
- void **setWidgetLookName** (const [String](#) &look)
- const [String](#) & **getWidgetNameSuffix** () const
- void **setWidgetNameSuffix** (const [String](#) &suffix)
- const [String](#) & **getWindowRendererType** () const
- void **setWindowRendererType** (const [String](#) &type)
- [VerticalAlignment](#) **getVerticalWidgetAlignment** () const
- void **setVerticalWidgetAlignment** ([VerticalAlignment](#) alignment)
- [HorizontalAlignment](#) **getHorizontalWidgetAlignment** () const
- void **setHorizontalWidgetAlignment** ([HorizontalAlignment](#) alignment)
- void **addPropertyInitialiser** (const [PropertyInitialiser](#) &initialiser)
- void **clearPropertyInitialisers** ()
- void **layout** (const [Window](#) &owner) const
- void **writeXMLToStream** ([XMLSerializer](#) &xml\_stream) const

*Writes an xml representation of this [WidgetComponent](#) to out\_stream.*

- const [PropertyInitialiser](#) \* **findPropertyInitialiser** (const [String](#) &propertyName) const

*Takes the name of a property and returns a pointer to the last [PropertyInitialiser](#) for this property or 0 if there is no [PropertyInitialiser](#) for this property in the [WidgetLookFeel](#).*

### 6.336.1 Detailed Description

Class that encapsulates information regarding a sub-widget required for a widget.

#### Todo

This is not finished in the slightest! There will be many changes here...

### 6.336.2 Member Function Documentation

#### 6.336.2.1 void CEGUI::WidgetComponent::writeXMLToStream (XMLSerializer & *xml\_stream*) const

Writes an xml representation of this [WidgetComponent](#) to *out\_stream*.

##### Parameters:

*xml\_stream* Stream where xml data should be output.

##### Returns:

Nothing.

#### 6.336.2.2 const PropertyInitialiser \* CEGUI::WidgetComponent::findPropertyInitialiser (const String & *propertyName*) const

Takes the name of a property and returns a pointer to the last [PropertyInitialiser](#) for this property or 0 if there is no [PropertyInitialiser](#) for this property in the [WidgetLookFeel](#).

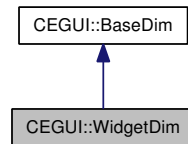
##### Parameters:

*propertyName* The name of the property to look for.

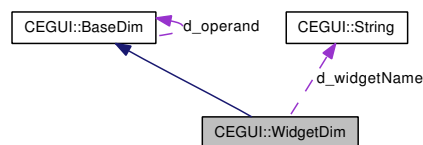
## 6.337 CEGUI::WidgetDim Class Reference

**Dimension** type that represents some dimension of a Window/widget. Implements **BaseDim** interface.

Inheritance diagram for CEGUI::WidgetDim:



Collaboration diagram for CEGUI::WidgetDim:



### Public Member Functions

- **WidgetDim** (const **String** &name, **DimensionType** dim)

*Constructor.*

- void **setWidgetName** (const **String** &name)

*Set the name suffix to use for this **WidgetDim**.*

- void **setSourceDimension** (**DimensionType** dim)

*Sets the source dimension type for this **WidgetDim**.*

### Protected Member Functions

- float **getValue\_impl** (const **Window** &wnd) const

*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically.*

- float **getValue\_impl** (const **Window** &wnd, const **Rect** &container) const

*Implementataion method to return the base value for this **BaseDim**. This method should not attempt to apply the mathematical operator; this is handled automatically by **BaseDim**.*

- void **writeXMLElementName\_impl** (**XMLSerializer** &xml\_stream) const

*Implementataion method to output real xml element name.*

- void **writeXMLElementAttributes\_impl** (**XMLSerializer** &xml\_stream) const

*Implementataion method to create the element attributes.*

- **BaseDim** \* **clone\_impl** () const



*Implementataion method to return a clone of this sub-class of [BaseDim](#). This method should not attempt to clone the mathematical operator or operand; theis is handled automatically by [BaseDim](#).*

### 6.337.1 Detailed Description

[Dimension](#) type that represents some dimension of a Window/widget. Implements [BaseDim](#) interface.

When calculating the final pixel value for the dimension, a target widget name is built by appending the name suffix specified for the [WidgetDim](#) to the name of the window passed to `getValue`, we then find the window/widget with that name - the final value of the dimension is taken from this window/widget.

### 6.337.2 Constructor & Destructor Documentation

#### 6.337.2.1 CEGUI::WidgetDim::WidgetDim (const String & *name*, DimensionType *dim*)

Constructor.

**Parameters:**

*name* [String](#) object holding the name suffix for a window/widget.

*dim* DimensionType value indicating which dimension of the described image that this [ImageDim](#) is to represent.

### 6.337.3 Member Function Documentation

#### 6.337.3.1 void CEGUI::WidgetDim::setWidgetName (const String & *name*)

Set the name suffix to use for this [WidgetDim](#).

**Parameters:**

*name* [String](#) object holding the name suffix for a window/widget.

**Returns:**

Nothing.

#### 6.337.3.2 void CEGUI::WidgetDim::setSourceDimension (DimensionType *dim*)

Sets the source dimension type for this [WidgetDim](#).

**Parameters:**

*dim* DimensionType value indicating which dimension of the described image that this [WidgetDim](#) is to represent.

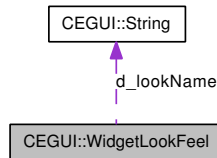
**Returns:**

Nothing.

## 6.338 CEGUI::WidgetLookAndFeel Class Reference

Class that encapsulates look & feel information for a particular widget type.

Collaboration diagram for CEGUI::WidgetLookAndFeel:



### Public Types

- typedef std::vector< [PropertyInitialiser](#) > [PropertyList](#)
- typedef std::vector< [PropertyDefinition](#) > [PropertyDefinitionList](#)
- typedef std::vector< [PropertyLinkDefinition](#) > [PropertyLinkDefinitionList](#)

### Public Member Functions

- **WidgetLookAndFeel** (const [String](#) &name)
- const [StateImagery](#) & [getStateImagery](#) (const [CEGUI::String](#) &state) const  
*Return a const reference to the [StateImagery](#) object for the specified state.*
- const [ImagerySection](#) & [getImagerySection](#) (const [CEGUI::String](#) &section) const  
*Return a const reference to the [ImagerySection](#) object with the specified name.*
- const [String](#) & [getName](#) () const  
*Return the name of the widget look.*
- void [addImagerySection](#) (const [ImagerySection](#) &section)  
*Add an [ImagerySection](#) to the [WidgetLookAndFeel](#).*
- void [addWidgetComponent](#) (const [WidgetComponent](#) &widget)  
*Add a [WidgetComponent](#) to the [WidgetLookAndFeel](#).*
- void [addStateSpecification](#) (const [StateImagery](#) &state)  
*Add a state specification ([StateImagery](#) object) to the [WidgetLookAndFeel](#).*
- void [addPropertyInitialiser](#) (const [PropertyInitialiser](#) &initialiser)  
*Add a property initialiser to the [WidgetLookAndFeel](#).*
- void [clearImagerySections](#) ()  
*Clear all [ImagerySections](#) from the [WidgetLookAndFeel](#).*
- void [clearWidgetComponents](#) ()  
*Clear all [WidgetComponents](#) from the [WidgetLookAndFeel](#).*
- void [clearStateSpecifications](#) ()

Clear all [StateImagery](#) objects from the [WidgetLookAndFeel](#).

- void [clearPropertyInitialisers](#) ()  
Clear all [PropertyInitialiser](#) objects from the [WidgetLookAndFeel](#).
- void [initialiseWidget](#) ([Window](#) &widget) const  
Initialise the given window using [PropertyInitialisers](#) and component widgets specified for this [WidgetLookAndFeel](#).
- void [cleanUpWidget](#) ([Window](#) &widget) const  
Clean up the given window from all properties and component widgets created by this [WidgetLookAndFeel](#).
- bool [isStateImageryPresent](#) (const [String](#) &state) const  
Return whether imagery is defined for the given state.
- void [addNamedArea](#) (const [NamedArea](#) &area)  
Adds a named area to the [WidgetLookAndFeel](#).
- void [clearNamedAreas](#) ()  
Clear all defined named areas from the [WidgetLookAndFeel](#).
- const [NamedArea](#) & [getNamedArea](#) (const [String](#) &name) const  
Return the [NamedArea](#) with the specified name.
- bool [isNamedAreaDefined](#) (const [String](#) &name) const  
return whether a [NamedArea](#) object with the specified name exists for this [WidgetLookAndFeel](#).
- void [layoutChildWidgets](#) (const [Window](#) &owner) const  
Layout the child widgets defined for this [WidgetLookAndFeel](#) which are attached to the given window.
- void [addPropertyDefinition](#) (const [PropertyDefinition](#) &propdef)  
Adds a property definition to the [WidgetLookAndFeel](#).
- void [addPropertyLinkDefinition](#) (const [PropertyLinkDefinition](#) &propdef)  
Adds a property link definition to the [WidgetLookAndFeel](#).
- void [clearPropertyDefinitions](#) ()  
Clear all defined property definitions from the [WidgetLookAndFeel](#).
- void [clearPropertyLinkDefinitions](#) ()  
Clear all defined property link definitions from the [WidgetLookAndFeel](#).
- void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const  
Writes an xml representation of this [WidgetLookAndFeel](#) to out\_stream.
- void [renameChildren](#) (const [Window](#) &widget, const [String](#) &newBaseName) const  
Uses the [WindowManager](#) to rename the child windows that are created for this [WidgetLookAndFeel](#).
- const [PropertyInitialiser](#) \* [findPropertyInitialiser](#) (const [String](#) &propertyName) const

*Takes the name of a property and returns a pointer to the last [PropertyInitialiser](#) for this property or 0 if there is no [PropertyInitialiser](#) for this property in the [WidgetLookFeel](#).*

- `const WidgetComponent * findWidgetComponent (const String &nameSuffix) const`  
*Takes the namesuffix for a widget component and returns a pointer to it if it exists or 0 if it does'nt.*
- `const PropertyDefinitionList & getPropertyDefinitions () const`
- `const PropertyLinkDefinitionList & getPropertyLinkDefinitions () const`
- `const PropertyList & getProperties () const`

### 6.338.1 Detailed Description

Class that encapsulates look & feel information for a particular widget type.

### 6.338.2 Member Typedef Documentation

#### 6.338.2.1 `typedef std::vector<PropertyInitialiser> CEGUI::WidgetLookFeel::PropertyList`

Typedefs for property related lists.

### 6.338.3 Member Function Documentation

#### 6.338.3.1 `const StateImagery & CEGUI::WidgetLookFeel::getStateImagery (const CEGUI::String & state) const`

Return a const reference to the [StateImagery](#) object for the specified state.

##### Returns:

[StateImagery](#) object for the requested state.

#### 6.338.3.2 `const ImagerySection & CEGUI::WidgetLookFeel::getImagerySection (const CEGUI::String & section) const`

Return a const reference to the [ImagerySection](#) object with the specified name.

##### Returns:

[ImagerySection](#) object with the specified name.

#### 6.338.3.3 `const String & CEGUI::WidgetLookFeel::getName () const`

Return the name of the widget look.

##### Returns:

[String](#) object holding the name of the [WidgetLookFeel](#).

**6.338.3.4 void CEGUI::WidgetLookAndFeel::addImagerySection (const ImagerySection & *section*)**

Add an [ImagerySection](#) to the [WidgetLookAndFeel](#).

**Parameters:**

*section* [ImagerySection](#) object to be added.

**Returns:**

Nothing.

**6.338.3.5 void CEGUI::WidgetLookAndFeel::addWidgetComponent (const WidgetComponent & *widget*)**

Add a [WidgetComponent](#) to the [WidgetLookAndFeel](#).

**Parameters:**

*widget* [WidgetComponent](#) object to be added.

**Returns:**

Nothing.

**6.338.3.6 void CEGUI::WidgetLookAndFeel::addStateSpecification (const StateImagery & *state*)**

Add a state specification ([StateImagery](#) object) to the [WidgetLookAndFeel](#).

**Parameters:**

*section* [StateImagery](#) object to be added.

**Returns:**

Nothing.

**6.338.3.7 void CEGUI::WidgetLookAndFeel::addPropertyInitialiser (const PropertyInitialiser & *initialiser*)**

Add a property initialiser to the [WidgetLookAndFeel](#).

**Parameters:**

*initialiser* [PropertyInitialiser](#) object to be added.

**Returns:**

Nothing.

**6.338.3.8 void CEGUI::WidgetLookAndFeel::clearImagerySections ()**

Clear all ImagerySections from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.9 void CEGUI::WidgetLookAndFeel::clearWidgetComponents ()**

Clear all WidgetComponents from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.10 void CEGUI::WidgetLookAndFeel::clearStateSpecifications ()**

Clear all [StateImagery](#) objects from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.11 void CEGUI::WidgetLookAndFeel::clearPropertyInitialisers ()**

Clear all [PropertyInitialiser](#) objects from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.12 void CEGUI::WidgetLookAndFeel::initialiseWidget (Window & *widget*) const**

Initialise the given window using PropertyInitialisers and component widgets specified for this [WidgetLookAndFeel](#).

**Parameters:**

*widget* [Window](#) based object to be initialised.

**Returns:**

Nothing.

**6.338.3.13 void CEGUI::WidgetLookAndFeel::cleanUpWidget (Window & *widget*) const**

Clean up the given window from all properties and component widgets created by this [WidgetLookAndFeel](#).

**Parameters:**

*widget* [Window](#) based object to be cleaned up.

**Returns:**

Nothing.

**6.338.3.14 bool CEGUI::WidgetLookAndFeel::isStateImageryPresent (const String & *state*) const**

Return whether imagery is defined for the given state.

**Parameters:**

*state* [String](#) object containing name of state to look for.

**Returns:**

- true if imagery exists for the specified state,
- false if no imagery exists for the specified state.

**6.338.3.15 void CEGUI::WidgetLookAndFeel::addNamedArea (const NamedArea & *area*)**

Adds a named area to the [WidgetLookAndFeel](#).

**Parameters:**

*area* [NamedArea](#) to be added.

**Returns:**

Nothing.

**6.338.3.16 void CEGUI::WidgetLookAndFeel::clearNamedAreas ()**

Clear all defined named areas from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.17 const NamedArea & CEGUI::WidgetLookAndFeel::getNamedArea (const String & *name*) const**

Return the [NamedArea](#) with the specified name.

**Parameters:**

*name* [String](#) object holding the name of the [NamedArea](#) to be returned.

**Returns:**

The requested [NamedArea](#) object.

**6.338.3.18 bool CEGUI::WidgetLookFeel::isNamedAreaDefined (const String & name) const**

return whether a [NamedArea](#) object with the specified name exists for this [WidgetLookFeel](#).

**Parameters:**

*name* [String](#) holding the name of the [NamedArea](#) to check for.

**Returns:**

- true if a named area with the requested name is defined for this [WidgetLookFeel](#).
- false if no such named area is defined for this [WidgetLookFeel](#).

**6.338.3.19 void CEGUI::WidgetLookFeel::layoutChildWidgets (const Window & owner) const**

Layout the child widgets defined for this [WidgetLookFeel](#) which are attached to the given window.

**Parameters:**

*owner* [Window](#) object that has the child widgets that require laying out.

**Returns:**

Nothing.

**6.338.3.20 void CEGUI::WidgetLookFeel::addPropertyDefinition (const PropertyDefinition & propdef)**

Adds a property definition to the [WidgetLookFeel](#).

**Parameters:**

*propdef* [PropertyDefinition](#) object to be added.

**Returns:**

Nothing.

**6.338.3.21 void CEGUI::WidgetLookFeel::addPropertyLinkDefinition (const PropertyLinkDefinition & propdef)**

Adds a property link definition to the [WidgetLookFeel](#).



**Parameters:**

*propdef* [PropertyLinkDefinition](#) object to be added.

**Returns:**

Nothing.

**6.338.3.22 void CEGUI::WidgetLookAndFeel::clearPropertyDefinitions ()**

Clear all defined property definitions from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.23 void CEGUI::WidgetLookAndFeel::clearPropertyLinkDefinitions ()**

Clear all defined property link definitions from the [WidgetLookAndFeel](#).

**Returns:**

Nothing.

**6.338.3.24 void CEGUI::WidgetLookAndFeel::writeXMLToStream (XMLSerializer & *xml\_stream*) const**

Writes an xml representation of this [WidgetLookAndFeel](#) to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.338.3.25 void CEGUI::WidgetLookAndFeel::renameChildren (const Window & *widget*, const String & *newBaseName*) const**

Uses the [WindowManager](#) to rename the child windows that are created for this [WidgetLookAndFeel](#).

**Parameters:**

*widget* The target [Window](#) containing the child windows that are to be renamed.

*newBaseName* [String](#) object holding the new base name that will be used when constructing new names for the child windows.

**6.338.3.26   const PropertyInitialiser \* CEGUI::WidgetLookFeel::findPropertyInitialiser (const String & *propertyName*) const**

Takes the name of a property and returns a pointer to the last [PropertyInitialiser](#) for this property or 0 if there is no [PropertyInitialiser](#) for this property in the [WidgetLookFeel](#).

**Parameters:**

*propertyName*   The name of the property to look for.

**6.338.3.27   const WidgetComponent \* CEGUI::WidgetLookFeel::findWidgetComponent (const String & *nameSuffix*) const**

Takes the namesuffix for a widget component and returns a pointer to it if it exists or 0 if it doesn't.

**Parameters:**

*nameSuffix*   The name suffix of the Child component to look for.

**6.338.3.28   const PropertyDefinitionList& CEGUI::WidgetLookFeel::getPropertyDefinitions () const   [inline]**

Obtains list of properties definitions. public

**Returns:**

CEGUI::WidgetLookFeel::PropertyDefinitionList List of properties definitions

**6.338.3.29   const PropertyLinkDefinitionList& CEGUI::WidgetLookFeel::getPropertyLinkDefinitions () const   [inline]**

Obtains list of properties link definitions. public

**Returns:**

CEGUI::WidgetLookFeel::PropertyLinkDefinitionList List of properties link definitions

**6.338.3.30   const PropertyList& CEGUI::WidgetLookFeel::getProperties () const   [inline]**

Obtains list of properties. public

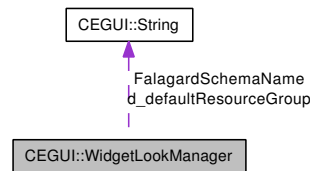
**Returns:**

[CEGUI::WidgetLookFeel::PropertyList](#) List of properties

## 6.339 CEGUI::WidgetLookManager Class Reference

Manager class that gives top-level access to widget data based "look and feel" specifications loaded into the system.

Collaboration diagram for CEGUI::WidgetLookManager:



### Public Member Functions

- [WidgetLookManager \(\)](#)  
*Constructor.*
- [~WidgetLookManager \(\)](#)  
*Destructor.*
- void [parseLookNFeelSpecification](#) (const [String](#) &filename, const [String](#) &resourceGroup="")  
*Parses a file containing window look & feel specifications (in the form of XML).*
- bool [isWidgetLookAvailable](#) (const [String](#) &widget) const  
*Return whether a [WidgetLookFeel](#) has been created with the specified name.*
- const [WidgetLookFeel](#) & [getWidgetLook](#) (const [String](#) &widget) const  
*Return a const reference to a [WidgetLookFeel](#) object which has the specified name.*
- void [eraseWidgetLook](#) (const [String](#) &widget)  
*Erase the [WidgetLookFeel](#) that has the specified name.*
- void [addWidgetLook](#) (const [WidgetLookFeel](#) &look)  
*Add the given [WidgetLookFeel](#).*
- void [writeWidgetLookToStream](#) (const [String](#) &name, [OutStream](#) &out\_stream) const  
*Writes a complete Widge Look to a stream. Note that xml file header and falagard opening/closing tags will also be written.*
- void [writeWidgetLookSeriesToStream](#) (const [String](#) &prefix, [OutStream](#) &out\_stream) const  
*Writes a series of complete Widge Look objects to a stream. Note that xml file header and falagard opening/closing tags will also be written.*

### Static Public Member Functions

- static [WidgetLookManager](#) & [getSingleton](#) (void)  
*Return singleton [WidgetLookManager](#) object.*

- static [WidgetLookManager](#) \* [getSingletonPtr](#) (void)  
*Return pointer to singleton [WindowFactoryManager](#) object.*
- static const [String](#) & [getDefaultResourceGroup](#) ()  
*Returns the default resource group currently set for LookNFeels.*
- static void [setDefaultResourceGroup](#) (const [String](#) &resourceGroup)  
*Sets the default resource group to be used when loading LookNFeel data.*

### 6.339.1 Detailed Description

Manager class that gives top-level access to widget data based "look and feel" specifications loaded into the system.

### 6.339.2 Member Function Documentation

#### 6.339.2.1 [WidgetLookManager](#) & [CEGUI::WidgetLookManager::getSingleton](#) (void) [static]

Return singleton [WidgetLookManager](#) object.

**Returns:**

Singleton [WidgetLookManager](#) object

#### 6.339.2.2 [WidgetLookManager](#) \* [CEGUI::WidgetLookManager::getSingletonPtr](#) (void) [static]

Return pointer to singleton [WindowFactoryManager](#) object.

**Returns:**

Pointer to singleton [WindowFactoryManager](#) object

#### 6.339.2.3 void [CEGUI::WidgetLookManager::parseLookNFeelSpecification](#) (const [String](#) & *filename*, const [String](#) & *resourceGroup* = "")

Parses a file containing window look & feel specifications (in the form of XML).

**Note:**

If the new file contains specifications for widget types that are already specified, it is not an error; the previous definitions are overwritten by the new data. An entry will appear in the log each time any look & feel component is overwritten.

**Parameters:**

*filename* [String](#) object containing the filename of a file containing the widget look & feel data

*resourceGroup* Resource group identifier to pass to the resource provider when loading the file.

**Returns:**

Nothing.

**Exceptions:**

*FileNotFoundException* thrown if there was some problem accessing or parsing the file *filename*

*InvalidRequestException* thrown if an invalid filename was provided.

#### 6.339.2.4 bool CEGUI::WidgetLookManager::isWidgetLookAvailable (const String & *widget*) const

Return whether a [WidgetLookFeel](#) has been created with the specified name.

**Parameters:**

*widget* [String](#) object holding the name of a widget look to test for.

**Returns:**

- true if a WindowLookFeel named *widget* is available.
- false if so such WindowLookFeel is currently available.

#### 6.339.2.5 const WidgetLookFeel & CEGUI::WidgetLookManager::getWidgetLook (const String & *widget*) const

Return a const reference to a [WidgetLookFeel](#) object which has the specified name.

**Parameters:**

*widget* [String](#) object holding the name of a widget look that is to be returned.

**Returns:**

const reference to the requested [WidgetLookFeel](#) object.

**Exceptions:**

*UnknownObjectException* thrown if no WindowLookFeel is available with the requested name.

#### 6.339.2.6 void CEGUI::WidgetLookManager::eraseWidgetLook (const String & *widget*)

Erase the [WidgetLookFeel](#) that has the specified name.

**Parameters:**

*widget* [String](#) object holding the name of a widget look to be erased. If no such WindowLookFeel exists, nothing happens.

**Returns:**

Nothing.

**6.339.2.7 void CEGUI::WidgetLookManager::addWidgetLook (const WidgetLookFeel & *look*)**

Add the given [WidgetLookFeel](#).

**Note:**

If the [WidgetLookFeel](#) specification uses a name that already exists within the system, it is not an error; the previous definition is overwritten by the new data. An entry will appear in the log each time any look & feel component is overwritten.

**Parameters:**

*look* [WidgetLookFeel](#) object to be added to the system. NB: The [WidgetLookFeel](#) is copied, no change of ownership of the input object occurs.

**Returns:**

Nothing.

**6.339.2.8 void CEGUI::WidgetLookManager::writeWidgetLookToStream (const String & *name*, OutStream & *out\_stream*) const**

Writes a complete Widget Look to a stream. Note that xml file header and falagard opening/closing tags will also be written.

**Parameters:**

*name* [String](#) holding the name of the widget look to be output to the stream.  
*out\_stream* OutStream where XML data should be sent.

**6.339.2.9 void CEGUI::WidgetLookManager::writeWidgetLookSeriesToStream (const String & *prefix*, OutStream & *out\_stream*) const**

Writes a series of complete Widget Look objects to a stream. Note that xml file header and falagard opening/closing tags will also be written.

The *prefix* specifies a name prefix common to all widget looks to be written, you could specify this as "TaharezLook/" and then any defined widget look starting with that prefix, such as "TaharezLook/Button" and "TaharezLook/Listbox" will be written to the stream.

**Parameters:**

*prefix* [String](#) holding the widget look name prefix, which will be used when searching for the widget looks to be output to the stream.  
*out\_stream* OutStream where XML data should be sent.

**6.339.2.10 static const String& CEGUI::WidgetLookManager::getDefaultResourceGroup (void) [inline, static]**

Returns the default resource group currently set for LookNFeels.

**Returns:**

[String](#) describing the default resource group identifier that will be used when loading LookNFeel data.

**6.339.2.11 static void CEGUI::WidgetLookManager::setDefaultResourceGroup (const String & *resourceGroup*) [inline, static]**

Sets the default resource group to be used when loading LookNFeel data.

**Parameters:**

*resourceGroup* [String](#) describing the default resource group identifier to be used.

**Returns:**

Nothing.

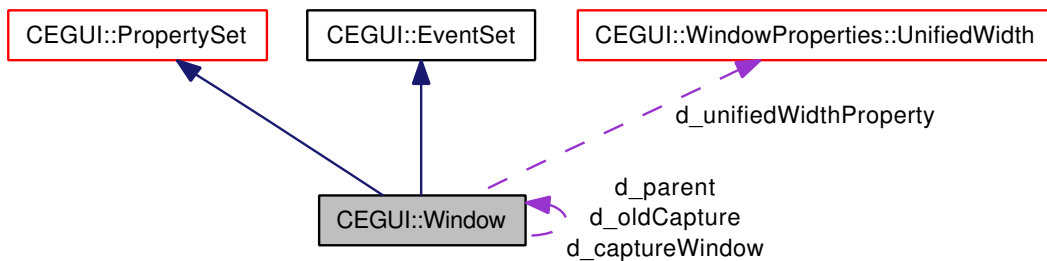
## 6.340 CEGUI::Window Class Reference

An abstract base class providing common functionality and specifying the required interface for derived classes.

Inherits [CEGUI::PropertySet](#), and [CEGUI::EventSet](#).

Inherited by [CEGUI::ButtonBase](#), [CEGUI::ClippedContainer](#), [CEGUI::Combobox](#), [CEGUI::DragContainer](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::GroupBox](#), [CEGUI::GUISheet](#), [CEGUI::ItemEntry](#), [CEGUI::ItemListBase](#), [CEGUI::Listbox](#), [CEGUI::ListHeader](#), [CEGUI::ListHeaderSegment](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::ProgressBar](#), [CEGUI::ScrollablePane](#), [CEGUI::Scrollbar](#), [CEGUI::ScrolledContainer](#), [CEGUI::Slider](#), [CEGUI::Spinner](#), [CEGUI::TabControl](#), [CEGUI::Titlebar](#), [CEGUI::Tooltip](#), and [CEGUI::Tree](#).

Collaboration diagram for CEGUI::Window:



### Public Member Functions

- [Window](#) (const [String](#) &type, const [String](#) &name)  
*Constructor for [Window](#) base class.*
- virtual [~Window](#) (void)  
*Destructor for [Window](#) base class.*
- const [String](#) & [getType](#) (void) const  
*return a [String](#) object holding the type name for this [Window](#).*
- const [String](#) & [getName](#) (void) const  
*return a [String](#) object holding the name of this [Window](#).*
- const [String](#) & [getPrefix](#) (void) const
- bool [isDestroyedByParent](#) (void) const  
*returns whether or not this [Window](#) is set to be destroyed when its parent window is destroyed.*
- bool [isAlwaysOnTop](#) (void) const  
*returns whether or not this [Window](#) is an always on top [Window](#). Also known as a top-most window.*
- bool [isDisabled](#) (bool localOnly=false) const  
*return whether the [Window](#) is currently disabled*
- bool [isVisible](#) (bool localOnly=false) const



return true if the [Window](#) is currently visible.

- bool [isActive](#) (void) const  
return true if this is the active [Window](#). An active window is a window that may receive user inputs.
- bool [isClippedByParent](#) (void) const  
return true if this [Window](#) is clipped so that its rendering will not pass outside of its parent [Window](#) area.
- uint [getID](#) (void) const  
return the ID code currently assigned to this [Window](#) by client code.
- size\_t [getChildCount](#) (void) const  
return the number of child [Window](#) objects currently attached to this [Window](#).
- bool [isChild](#) (const [String](#) &name) const  
returns whether a [Window](#) with the specified name is currently attached to this [Window](#) as a child.
- bool [isChild](#) (uint ID) const  
returns whether at least one window with the given ID code is attached to this [Window](#) as a child.
- bool [isChildRecursive](#) (uint ID) const  
returns whether at least one window with the given ID code is attached to this [Window](#) or any of it's children as a child.
- bool [isChild](#) (const [Window](#) \*window) const  
return true if the given [Window](#) is a child of this window.
- [Window](#) \* [getChild](#) (const [String](#) &name) const  
return a pointer to the child window with the specified name.
- [Window](#) \* [recursiveChildSearch](#) (const [String](#) &name) const
- [Window](#) \* [getChild](#) (uint ID) const  
return a pointer to the first attached child window with the specified ID value.
- [Window](#) \* [getChildRecursive](#) (uint ID) const  
return a pointer to the first attached child window with the specified ID value. Children are traversed recursively.
- [Window](#) \* [getChildAtIndex](#) (size\_t idx) const  
return a pointer to the child window that is attached to 'this' at the given index.
- [Window](#) \* [getActiveChild](#) (void)  
return a pointer to the [Window](#) that currently has input focus starting with this [Window](#).
- const [Window](#) \* [getActiveChild](#) (void) const
- bool [isAncestor](#) (const [String](#) &name) const  
return true if the specified [Window](#) is some ancestor of this [Window](#)
- bool [isAncestor](#) (uint ID) const  
return true if any [Window](#) with the given ID is some ancestor of this [Window](#).

- `bool isAncestor (const Window *window) const`  
*return true if the specified Window is some ancestor of this Window.*
- `Font * getFont (bool useDefault=true) const`  
*return the active Font object for the Window.*
- `const String & getText (void) const`  
*return the current text for the Window*
- `bool inheritsAlpha (void) const`  
*return true if the Window inherits alpha from its parent(s).*
- `float getAlpha (void) const`  
*return the current alpha value set for this Window*
- `float getEffectiveAlpha (void) const`  
*return the effective alpha value that will be used when rendering this window, taking into account inheritance of parent window(s) alpha.*
- `Rect getPixelRect (void) const`  
*return a Rect object describing the Window area in screen space.*
- `virtual Rect getPixelRect_impl (void) const`  
*return a Rect object describing the Window area in screen space.*
- `Rect getInnerRect (void) const`  
*return a Rect object describing the clipped inner area for this window.*
- `Rect getUnclippedPixelRect (void) const`  
*return a Rect object describing the Window area unclipped, in screen space.*
- `Rect getUnclippedInnerRect (void) const`  
*Return a Rect object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.*
- `virtual Rect getUnclippedInnerRect_impl (void) const`  
*Return a Rect object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.*
- `bool isCapturedByThis (void) const`  
*return true if this Window has input captured.*
- `bool isCapturedByAncestor (void) const`  
*return true if an ancestor window has captured inputs.*
- `bool isCapturedByChild (void) const`  
*return true if a child window has captured inputs.*

- virtual bool **isHit** (const **Vector2** &position) const  
*check if the given pixel position would hit this window.*
- **Window** \* **getChildAtPosition** (const **Vector2** &position) const  
*return the child **Window** that is hit by the given pixel position*
- **Window** \* **getTargetChildAtPosition** (const **Vector2** &position) const  
*return the child **Window** that is 'hit' by the given position, and is allowed to handle mouse events.*
- **Window** \* **getParent** (void) const  
*return the parent of this **Window**.*
- const **Image** \* **getMouseCursor** (bool useDefault=true) const  
*Return a pointer to the mouse cursor image to use when the mouse cursor is within this window's area.*
- **Size** **getPixelSize** (void) const  
*Return the window size in pixels.*
- void \* **getUserData** (void) const  
*Return the user data set for this **Window**.*
- bool **restoresOldCapture** (void) const  
*Return whether this window is set to restore old input capture when it loses input capture.*
- bool **isZOrderingEnabled** (void) const  
*Return whether z-order changes are enabled or disabled for this **Window**.*
- bool **wantsMultiClickEvents** (void) const  
*Return whether this window will receive multi-click events or multiple 'down' events instead.*
- bool **isMouseAutoRepeatEnabled** (void) const  
*Return whether mouse button down event autorepeat is enabled for this window.*
- float **getAutoRepeatDelay** (void) const  
*Return the current auto-repeat delay setting for this window.*
- float **getAutoRepeatRate** (void) const  
*Return the current auto-repeat rate setting for this window.*
- bool **distributesCapturedInputs** (void) const  
*Return whether the window wants inputs passed to its attached child windows when the window has inputs captured.*
- bool **isUsingDefaultTooltip** (void) const  
*Return whether this **Window** is using the system default **Tooltip** for its **Tooltip** window.*
- **Tooltip** \* **getTooltip** (void) const  
*Return a pointer to the **Tooltip** object used by this **Window**. The value returned may point to the system default **Tooltip**, a custom **Window** specific **Tooltip**, or be NULL.*

- [String](#) [getTooltipType](#) (void) const  
*Return the custom tooltip type.*
- const [String](#) & [getTooltipText](#) (void) const  
*Return the current tooltip text set for this [Window](#).*
- bool [inheritsTooltipText](#) (void) const  
*Return whether this window inherits [Tooltip](#) text from its parent when its own tooltip text is not set.*
- bool [isRaiseOnClickEnabled](#) (void) const  
*Return whether this window will rise to the top of the z-order when clicked with the left mouse button.*
- bool [testClassName](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- [VerticalAlignment](#) [getVerticalAlignment](#) () const  
*Get the vertical alignment.*
- [HorizontalAlignment](#) [getHorizontalAlignment](#) () const  
*Get the horizontal alignment.*
- [RenderCache](#) & [getRenderCache](#) ()  
*Return the [RenderCache](#) object for this [Window](#).*
- const [String](#) & [getLookNFeel](#) () const  
*Get the name of the LookNFeel assigned to this window.*
- bool [getModalState](#) (void) const  
*Get whether or not this [Window](#) is the modal target.*
- const [String](#) & [getUserString](#) (const [String](#) &name) const  
*Returns a named user string.*
- bool [isUserStringDefined](#) (const [String](#) &name) const  
*Return whether a user string with the specified name exists.*
- [Window](#) \* [getActiveSibling](#) ()  
*Returns the active sibling window.*
- [Size](#) [getParentPixelSize](#) (void) const  
*Return the pixel size of the parent element. This always returns a valid object.*
- float [getParentPixelWidth](#) (void) const  
*Return the pixel Width of the parent element. This always returns a valid number.*
- float [getParentPixelHeight](#) (void) const  
*Return the pixel Height of the parent element. This always returns a valid number.*
- bool [isMousePassThroughEnabled](#) (void) const

*Returns whether this window should ignore mouse event and pass them through to and other windows behind it. In effect making the window transparent to the mouse.*

- bool [isAutoWindow](#) (void) const

*Returns whether this window is an auto-child window. All auto-child windows have "\_\_auto\_" in their name, but this is faster.*

- bool [isWritingXMLAllowed](#) (void) const

*Returns whether this window is allowed to write XML.*

- [EventSet::Iterator](#) [getEventIterator](#) () const

*Helper method that returns an [EventSet::Iterator](#) object that can be used to iterate over the events currently added to the [EventSet](#) of this [Window](#).*

- [PropertySet::Iterator](#) [getPropertyIterator](#) () const

*Helper method that returns a [PropertySet::Iterator](#) object that can be used to iterate over the events currently added to the [PropertySet](#) of this [Window](#).*

- bool [isDragDropTarget](#) () const

*Returns whether this [Window](#) object will receive events generated by the drag and drop support in the system.*

- void [rename](#) (const [String](#) &new\_name)

*Renames the window.*

- virtual void [initialiseComponents](#) (void)

*Initialises the [Window](#) based object ready for use.*

- void [setDestroyedByParent](#) (bool setting)

*Set whether or not this [Window](#) will automatically be destroyed when its parent [Window](#) is destroyed.*

- void [setAlwaysOnTop](#) (bool setting)

*Set whether this window is always on top, or not.*

- void [setEnabled](#) (bool setting)

*Set whether this window is enabled or disabled. A disabled window normally can not be interacted with, and may have different rendering.*

- void [enable](#) (void)

*enable the [Window](#) to allow interaction.*

- void [disable](#) (void)

*disable the [Window](#) to prevent interaction.*

- void [setVisible](#) (bool setting)

*Set whether the [Window](#) is visible or hidden.*

- void [show](#) (void)

*show the [Window](#)*

- void [hide](#) (void)

hide the [Window](#).

- void [activate](#) (void)

Activate the [Window](#) giving it input focus and bringing it to the top of all windows with the same always-on-top setting as this [Window](#).

- void [deactivate](#) (void)

Deactivate the window. No further inputs will be received by the window until it is re-activated either programmatically or by the user interacting with the gui.

- void [setClippedByParent](#) (bool setting)

Set whether this [Window](#) will be clipped by its parent window(s).

- void [setID](#) (uint ID)

Set the current ID for the [Window](#).

- void [setPrefix](#) (String prefix)

- void [setText](#) (const String &text)

Set the current text string for the [Window](#).

- void [setFont](#) (Font \*font)

Set the font used by this [Window](#).

- void [setFont](#) (const String &name)

Set the font used by this [Window](#).

- void [addChildWindow](#) (const String &name)

Add the named [Window](#) as a child of this [Window](#). If the [Window](#) name is already attached to a [Window](#), it is detached before being added to this [Window](#).

- void [addChildWindow](#) (Window \*window)

Add the specified [Window](#) as a child of this [Window](#). If the [Window](#) window is already attached to a [Window](#), it is detached before being added to this [Window](#).

- void [removeChildWindow](#) (const String &name)

Remove the named [Window](#) from this windows child list.

- void [removeChildWindow](#) (Window \*window)

Remove the specified [Window](#) from this windows child list.

- void [removeChildWindow](#) (uint ID)

Remove the first child [Window](#) with the specified ID. If there is more than one attached [Window](#) objects with the specified ID, only the first one encountered will be removed.

- void [moveToFront](#) ()

Move the [Window](#) to the top of the z order.

- void [moveToBack](#) ()

Move the [Window](#) to the bottom of the Z order.

- bool [captureInput](#) (void)

*Captures input to this window.*

- void [releaseInput](#) (void)  
*Releases input capture from this [Window](#). If this [Window](#) does not have inputs captured, nothing happens.*
- void [setRestoreCapture](#) (bool setting)  
*Set whether this window will remember and restore the previous window that had inputs captured.*
- void [setAlpha](#) (float alpha)  
*Set the current alpha value for this window.*
- void [setInheritsAlpha](#) (bool setting)  
*Sets whether this [Window](#) will inherit alpha from its parent windows.*
- void [requestRedraw](#) (void) const  
*Signal the [System](#) object to redraw (at least) this [Window](#) on the next render cycle.*
- void [setMouseCursor](#) (const [Image](#) \*image)  
*Set the mouse cursor image to be used when the mouse enters this window.*
- void [setMouseCursor](#) ([MouseCursorImage](#) image)  
*Set the mouse cursor image to be used when the mouse enters this window.*
- void [setMouseCursor](#) (const [String](#) &imageset, const [String](#) &image\_name)  
*Set the mouse cursor image to be used when the mouse enters this window.*
- void [setUserData](#) (void \*user\_data)  
*Set the user data set for this [Window](#).*
- void [setZOrderingEnabled](#) (bool setting)  
*Set whether z-order changes are enabled or disabled for this [Window](#).*
- void [setWantsMultiClickEvents](#) (bool setting)  
*Set whether this window will receive multi-click events or multiple 'down' events instead.*
- void [setMouseAutoRepeatEnabled](#) (bool setting)  
*Set whether mouse button down event autorepeat is enabled for this window.*
- void [setAutoRepeatDelay](#) (float delay)  
*Set the current auto-repeat delay setting for this window.*
- void [setAutoRepeatRate](#) (float rate)  
*Set the current auto-repeat rate setting for this window.*
- void [setDistributesCapturedInputs](#) (bool setting)  
*Set whether the window wants inputs passed to its attached child windows when the window has inputs captured.*
- void [notifyDragDropItemEnters](#) ([DragContainer](#) \*item)

*Internal support method for drag & drop. You do not normally call this directly from client code. See the [DragContainer](#) class.*

- void [notifyDragDropItemLeaves](#) ([DragContainer](#) \*item)  
*Internal support method for drag & drop. You do not normally call this directly from client code. See the [DragContainer](#) class.*
- void [notifyDragDropItemDropped](#) ([DragContainer](#) \*item)  
*Internal support method for drag & drop. You do not normally call this directly from client code. See the [DragContainer](#) class.*
- virtual void [destroy](#) (void)  
*Internal destroy method which actually just adds the window and any parent destructed child windows to the dead pool.*
- void [setTooltip](#) ([Tooltip](#) \*tooltip)  
*Set the custom [Tooltip](#) object for this [Window](#). This value may be 0 to indicate that the [Window](#) should use the system default [Tooltip](#) object.*
- void [setTooltipType](#) (const [String](#) &tooltipType)  
*Set the custom [Tooltip](#) to be used by this [Window](#) by specifying a [Window](#) type.*
- void [setTooltipText](#) (const [String](#) &tip)  
*Set the tooltip text for this window.*
- void [setInheritsTooltipText](#) (bool setting)  
*Set whether this window inherits [Tooltip](#) text from its parent when its own tooltip text is not set.*
- void [setRiseOnClickEnabled](#) (bool setting)  
*Set whether this window will rise to the top of the z-order when clicked with the left mouse button.*
- void [setVerticalAlignment](#) (const [VerticalAlignment](#) alignment)  
*Set the vertical alignment.*
- void [setHorizontalAlignment](#) (const [HorizontalAlignment](#) alignment)  
*Set the horizontal alignment.*
- virtual void [setLookNFeel](#) (const [String](#) &look)  
*Set the [LookNFeel](#) that should be used for this window.*
- void [setModalState](#) (bool state)  
*Set the modal state for this [Window](#).*
- virtual void [performChildWindowLayout](#) ()  
*method called to perform extended laying out of attached child windows.*
- void [setUserString](#) (const [String](#) &name, const [String](#) &value)  
*Sets the value a named user string, creating it as required.*
- void [setArea](#) (const [UDim](#) &xpos, const [UDim](#) &ypos, const [UDim](#) &width, const [UDim](#) &height)  
*Set the window area.*



- void [setArea](#) (const [UVector2](#) &pos, const [UVector2](#) &size)  
*Set the window area.*
- void [setArea](#) (const [URect](#) &area)  
*Set the window area.*
- void [setPosition](#) (const [UVector2](#) &pos)  
*Set the window's position.*
- void [setXPosition](#) (const [UDim](#) &x)  
*Set the window's X position.*
- void [setYPosition](#) (const [UDim](#) &y)  
*Set the window's Y position.*
- void [setSize](#) (const [UVector2](#) &size)  
*Set the window's size.*
- void [setWidth](#) (const [UDim](#) &width)  
*Set the window's width.*
- void [setHeight](#) (const [UDim](#) &height)  
*Set the window's height.*
- void [setMaxSize](#) (const [UVector2](#) &size)  
*Set the window's maximum size.*
- void [setMinSize](#) (const [UVector2](#) &size)  
*Set the window's minimum size.*
- const [URect](#) & [getArea](#) () const  
*Return the windows area.*
- const [UVector2](#) & [getPosition](#) () const  
*Get the window's position.*
- const [UDim](#) & [getXPosition](#) () const  
*Get the window's X position.*
- const [UDim](#) & [getYPosition](#) () const  
*Get the window's Y position.*
- [UVector2](#) [getSize](#) () const  
*Get the window's size.*
- [UDim](#) [getWidth](#) () const  
*Get the window's width.*
- [UDim](#) [getHeight](#) () const

*Get the window's height.*

- const [UVector2](#) & [getMaxSize](#) () const

*Get the window's maximum size.*

- const [UVector2](#) & [getMinSize](#) () const

*Get the window's minimum size.*

- void [render](#) (void)

*Causes the [Window](#) object to render itself and all of it's attached children.*

- void [update](#) (float elapsed)

*Cause window to update itself and any attached children. Client code does not need to call this method; to ensure full, and proper updates, call the [injectTimePulse](#) methodname method provided by the [System](#) class.*

- virtual void [writeXMLToStream](#) ([XMLSerializer](#) &xml\_stream) const

*Writes an xml representation of this window object to out\_stream.*

- virtual void [beginInitialisation](#) (void)

*Sets the internal 'initialising' flag to true. This can be use to optimize initialisation of some widgets, and is called automatically by the layout XML handler when it has created a window. That is just after the window has been created, but before any children or properties are read.*

- virtual void [endInitialisation](#) (void)

*Sets the internal 'initialising' flag to false. This is called automatically by the layout XML handler when it is done creating a window. That is after all properties and children have been loaded and just before the next sibling gets created.*

- void [setMousePassThroughEnabled](#) (bool setting)

*Sets whether this window should ignore mouse events and pass them through to any windows behind it. In effect making the window transparent to the mouse.*

- void [setWindowRenderer](#) (const [String](#) &name)

*Assign the [WindowRenderer](#) to specify the Look'N'Feel specification to be used.*

- [WindowRenderer](#) \* [getWindowRenderer](#) (void) const

*Get the currently assigned [WindowRenderer](#). (Look'N'Feel specification).*

- [String](#) [getWindowRendererName](#) (void) const

*Get the factory name of the currently assigned [WindowRenderer](#). (Look'N'Feel specification).*

- void [setWritingXMLAllowed](#) (bool allow)

*Sets whether this window is allowed to write XML.*

- void [notifyScreenAreaChanged](#) (void)

*Recursively inform all children that the screen area has changed, and needs to be re-cached.*

- void [setFalagardType](#) (const [String](#) &type, const [String](#) &rendererType="")

*Changes the widget's falagard type, thus changing its look'n'feel and optionally its renderer in the process.*

- void [setDragDropTarget](#) (bool setting)

*Specifies whether this [Window](#) object will receive events generated by the drag and drop support in the system.*

## Static Public Member Functions

- static [Window](#) \* [getCaptureWindow](#) (void)  
*return the [Window](#) that currently has inputs captured.*

## Static Public Attributes

- static const [String](#) [EventNamespace](#)  
*< Namespace for global events*
- static const [String](#) [EventWindowUpdated](#)
- static const [String](#) [EventParentSized](#)  
*Parent of this [Window](#) has been re-sized.*
- static const [String](#) [EventSized](#)  
*[Window](#) size has changed.*
- static const [String](#) [EventMoved](#)  
*[Window](#) position has changed.*
- static const [String](#) [EventTextChanged](#)  
*Text string for the [Window](#) has changed.*
- static const [String](#) [EventFontChanged](#)  
*Font object for the [Window](#) has been changed.*
- static const [String](#) [EventAlphaChanged](#)  
*Alpha blend value for the [Window](#) has changed.*
- static const [String](#) [EventIDChanged](#)  
*Client assigned ID code for the [Window](#) has changed.*
- static const [String](#) [EventActivated](#)  
*[Window](#) has been activated (has input focus).*
- static const [String](#) [EventDeactivated](#)  
*[Window](#) has been deactivated (loses input focus).*
- static const [String](#) [EventShown](#)  
*[Window](#) has been made visible.*
- static const [String](#) [EventHidden](#)  
*[Window](#) has been hidden from view.*

- static const [String EventEnabled](#)  
*Window has been enabled (interaction is possible).*
- static const [String EventDisabled](#)  
*Window has been disabled (interaction is no longer possible).*
- static const [String EventClippedByParentChanged](#)  
*Clipping by parent mode has been modified.*
- static const [String EventDestroyedByParentChanged](#)  
*Destruction by parent mode has been modified.*
- static const [String EventInheritsAlphaChanged](#)  
*Alpha inherited from parent mode has been modified.*
- static const [String EventAlwaysOnTopChanged](#)  
*Always on top mode has been modified.*
- static const [String EventInputCaptureGained](#)  
*Window has captured all inputs.*
- static const [String EventInputCaptureLost](#)  
*Window has lost it's capture on inputs.*
- static const [String EventRenderingStarted](#)  
*Rendering of the Window has started.*
- static const [String EventRenderingEnded](#)  
*Rendering for the Window has finished.*
- static const [String EventChildAdded](#)  
*A child Window has been added.*
- static const [String EventChildRemoved](#)  
*A child window has been removed.*
- static const [String EventDestructionStarted](#)  
*Destruction of the Window is about to begin.*
- static const [String EventZOrderChanged](#)  
*The z-order of the window has changed.*
- static const [String EventDragDropItemEnters](#)  
*A DragContainer has been dragged over this window.*
- static const [String EventDragDropItemLeaves](#)  
*A DragContainer has left this window.*
- static const [String EventDragDropItemDropped](#)  
*A DragContainer was dropped on this Window.*

- static const [String EventVerticalAlignmentChanged](#)  
*The vertical alignment of the window has changed.*
- static const [String EventHorizontalAlignmentChanged](#)  
*The vertical alignment of the window has changed.*
- static const [String EventWindowRendererAttached](#)  
*The a new window renderer was attached.*
- static const [String EventWindowRendererDetached](#)  
*The currently assigned window renderer was detached.*
- static const [String EventMouseEnters](#)  
*Mouse cursor has entered the [Window](#).*
- static const [String EventMouseLeaves](#)  
*Mouse cursor has left the [Window](#).*
- static const [String EventMouseMove](#)  
*Mouse cursor was moved within the area of the [Window](#).*
- static const [String EventMouseWheel](#)  
*Mouse wheel was scrolled within the [Window](#).*
- static const [String EventMouseButtonDown](#)  
*A mouse button was pressed down within the [Window](#).*
- static const [String EventMouseButtonUp](#)  
*A mouse button was released within the [Window](#).*
- static const [String EventMouseClicked](#)  
*A mouse button was clicked (down then up) within the [Window](#).*
- static const [String EventMouseDoubleClick](#)  
*A mouse button was double-clicked within the [Window](#).*
- static const [String EventMouseTripleClick](#)  
*A mouse button was triple-clicked within the [Window](#).*
- static const [String EventKeyDown](#)  
*A key on the keyboard was pressed.*
- static const [String EventKeyUp](#)  
*A key on the keyboard was released.*
- static const [String EventCharacterKey](#)  
*A text character was typed on the keyboard.*
- static const [String TooltipNameSuffix](#)

*Widget name suffix for automatically created tooltip widgets.*

- static const [String](#) [AutoWidgetNameSuffix](#)

*Something that all generated widgets will have in their names.*

## Protected Types

- typedef std::vector< [Window](#) \* > [ChildList](#)
- typedef std::map< [String](#), [String](#), [String::FastLessCompare](#) > [UserStringMap](#)
- typedef std::set< [String](#), [String::FastLessCompare](#) > [BannedXMLPropertySet](#)

*std::set used to determine whether a window should write a property to XML or not. if the property name is present the property will not be written*

## Protected Member Functions

- virtual void [onSized](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's size changes.*
- virtual void [onMoved](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's position changes.*
- virtual void [onTextChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's text is changed.*
- virtual void [onFontChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's font is changed.*
- virtual void [onAlphaChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's alpha blend value is changed.*
- virtual void [onIDChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's client assigned ID is changed.*
- virtual void [onShown](#) ([WindowEventArgs](#) &e)  
*Handler called when the window is shown (made visible).*
- virtual void [onHidden](#) ([WindowEventArgs](#) &e)  
*Handler called when the window is hidden.*
- virtual void [onEnabled](#) ([WindowEventArgs](#) &e)  
*Handler called when the window is enabled.*
- virtual void [onDisabled](#) ([WindowEventArgs](#) &e)  
*Handler called when the window is disabled.*
- virtual void [onClippingChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's setting for being clipped by it's parent is changed.*

- virtual void [onParentDestroyChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's setting for being destroyed automatically by its parent is changed.*
- virtual void [onInheritsAlphaChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's setting for inheriting alpha-blending is changed.*
- virtual void [onAlwaysOnTopChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the window's always-on-top setting is changed.*
- virtual void [onCaptureGained](#) ([WindowEventArgs](#) &e)  
*Handler called when this window gains capture of mouse inputs.*
- virtual void [onCaptureLost](#) ([WindowEventArgs](#) &e)  
*Handler called when this window loses capture of mouse inputs.*
- virtual void [onRenderingStarted](#) ([WindowEventArgs](#) &e)  
*Handler called when rendering for this window has started.*
- virtual void [onRenderingEnded](#) ([WindowEventArgs](#) &e)  
*Handler called when rendering for this window has ended.*
- virtual void [onZChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the z-order position of this window has changed.*
- virtual void [onDestructionStarted](#) ([WindowEventArgs](#) &e)  
*Handler called when this window's destruction sequence has begun.*
- virtual void [onActivated](#) ([ActivationEventArgs](#) &e)  
*Handler called when this window has become the active window.*
- virtual void [onDeactivated](#) ([ActivationEventArgs](#) &e)  
*Handler called when this window has lost input focus and has been deactivated.*
- virtual void [onParentSized](#) ([WindowEventArgs](#) &e)  
*Handler called when this window's parent window has been resized. If this window is the root / GUI Sheet window, this call will be made when the display size changes.*
- virtual void [onChildAdded](#) ([WindowEventArgs](#) &e)  
*Handler called when a child window is added to this window.*
- virtual void [onChildRemoved](#) ([WindowEventArgs](#) &e)  
*Handler called when a child window is removed from this window.*
- virtual void [onMouseEnters](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has entered this window's area.*
- virtual void [onMouseLeaves](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has left this window's area.*

- virtual void [onMouseMove](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse cursor has been moved within this window's area.*
- virtual void [onMouseWheel](#) ([MouseEventArgs](#) &e)  
*Handler called when the mouse wheel (z-axis) position changes within this window's area.*
- virtual void [onMouseDown](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been depressed within this window's area.*
- virtual void [onMouseUp](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been released within this window's area.*
- virtual void [onMouseClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been clicked (that is depressed and then released, within a specified time) within this window's area.*
- virtual void [onMouseDoubleClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been double-clicked within this window's area.*
- virtual void [onMouseTripleClicked](#) ([MouseEventArgs](#) &e)  
*Handler called when a mouse button has been triple-clicked within this window's area.*
- virtual void [onKeyDown](#) ([KeyEventArgs](#) &e)  
*Handler called when a key as been depressed while this window has input focus.*
- virtual void [onKeyUp](#) ([KeyEventArgs](#) &e)  
*Handler called when a key as been released while this window has input focus.*
- virtual void [onCharacter](#) ([KeyEventArgs](#) &e)  
*Handler called when a character-key has been pressed while this window has input focus.*
- virtual void [onDragDropItemEnters](#) ([DragDropEventArgs](#) &e)  
*Handler called when a [DragContainer](#) is dragged over this window.*
- virtual void [onDragDropItemLeaves](#) ([DragDropEventArgs](#) &e)  
*Handler called when a [DragContainer](#) is dragged over this window.*
- virtual void [onDragDropItemDropped](#) ([DragDropEventArgs](#) &e)  
*Handler called when a [DragContainer](#) is dragged over this window.*
- virtual void [onVerticalAlignmentChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the vertical alignment setting for the window is changed.*
- virtual void [onHorizontalAlignmentChanged](#) ([WindowEventArgs](#) &e)  
*Handler called when the horizontal alignment setting for the window is changed.*
- virtual void [onWindowRendererAttached](#) ([WindowEventArgs](#) &e)  
*Handler called when a new window renderer object is attached.*
- virtual void [onWindowRendererDetached](#) ([WindowEventArgs](#) &e)



*Handler called when the currently attached window renderer object is detached.*

- virtual void [updateSelf](#) (float elapsed)  
*Perform actual update processing for this [Window](#).*
- virtual void [drawSelf](#) (float z)  
*Perform the actual rendering for this [Window](#).*
- virtual void [populateRenderCache](#) ()  
*Update the rendering cache.*
- virtual bool [testClassName\\_impl](#) (const [String](#) &class\_name) const  
*Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.*
- void [setParent](#) ([Window](#) \*parent)  
*Set the parent window for this window object.*
- [Size](#) [getSize\\_impl](#) (const [Window](#) \*window) const
- void [generateAutoRepeatEvent](#) ([MouseButton](#) button)  
*Fires off a repeated mouse button down event for this window.*
- virtual bool [validateWindowRenderer](#) (const [String](#) &name) const  
*Function used in checking if a [WindowRenderer](#) is valid for this window.*
- void [banPropertyFromXML](#) (const [Property](#) \*property)  
*Adds a property to the XML ban list.*
- bool [isPropertyBannedFromXML](#) (const [Property](#) \*property) const  
*Returns whether a property is banned from XML.*
- bool [isPropertyAtDefault](#) (const [Property](#) \*property) const  
*Returns whether a property is at it's default value. This function is different from [Property::isDefault](#) as it takes the assigned look'n'feel (if there is one) into account.*
- void [notifyClippingChanged](#) (void)  
*Recursively inform all children that the clipping has changed and screen rects need to be recalculated.*
- virtual void [cleanupChildren](#) (void)  
*Cleanup child windows.*
- virtual void [addChild\\_impl](#) ([Window](#) \*wnd)  
*Add given window to child list at an appropriate position.*
- virtual void [removeChild\\_impl](#) ([Window](#) \*wnd)  
*Remove given window from child list.*
- virtual void [onZChange\\_impl](#) (void)  
*Notify 'this' and all siblings of a ZOrder change event.*

- void [addStandardProperties](#) (void)  
*Add standard [CEGUI::Window](#) properties.*
- virtual bool [moveToFront\\_impl](#) (bool wasClicked)  
*Implements move to front behavior.*
- bool [doRiseOnClick](#) (void)  
*Implementation of rise on click functionality.*
- void [setArea\\_impl](#) (const [UVector2](#) &pos, const [UVector2](#) &size, bool topLeftSizing=false, bool fireEvents=true)  
*Implementation method to modify window area while correctly applying min / max size processing, and firing any appropriate events.*
- void [addWindowToDrawList](#) ([Window](#) &wnd, bool at\_back=false)  
*Add the given window to the drawing list at an appropriate position for it's settings and the required direction. Basically, when at\_back is false, the window will appear in front of all other windows with the same 'always on top' setting. When at\_back is true, the window will appear behind all other windows with the same 'always on top' setting.*
- void [removeWindowFromDrawList](#) (const [Window](#) &wnd)  
*Removes the window from the drawing list. If the window is not attached to the drawing list then nothing happens.*
- bool [isTopOfZOrder](#) () const  
*Return whether the window is at the top of the Z-Order. This will correctly take into account 'Always on top' windows as needed.*
- virtual int [writePropertiesXML](#) ([XMLSerializer](#) &xml\_stream) const
- virtual int [writeChildWindowsXML](#) ([XMLSerializer](#) &xml\_stream) const
- virtual bool [writeAutoChildWindowXML](#) ([XMLSerializer](#) &xml\_stream) const
- [Window](#) (const [Window](#) &wnd)
- [Window](#) & [operator=](#) (const [Window](#) &wnd)

## Protected Attributes

- ChildList [d\\_children](#)  
*The list of child [Window](#) objects attached to this.*
- ChildList [d\\_drawList](#)  
*Child window objects arranged in rendering order.*
- [Window](#) \* [d\\_oldCapture](#)  
*The [Window](#) that previously had capture (used for restoreOldCapture mode).*
- [Window](#) \* [d\\_parent](#)  
*Holds pointer to the parent window.*
- [Font](#) \* [d\\_font](#)  
*Holds pointer to the [Window](#) objects current [Font](#).*

- [String d\\_text](#)  
*Holds the text / label / caption for this [Window](#).*
- [uint d\\_ID](#)  
*User ID assigned to this [Window](#).*
- [float d\\_alpha](#)  
*Alpha transparency setting for the [Window](#).*
- [URect d\\_area](#)  
*This [Window](#) objects area as defined by a [URect](#).*
- [Size d\\_pixelSize](#)  
*Current constrained pixel size of the window.*
- [const Image \\* d\\_mouseCursor](#)  
*Holds pointer to the [Window](#) objects current mouse cursor image.*
- [UserStringMap d\\_userStrings](#)  
*Holds a collection of named user string values.*
- [void \\* d\\_userData](#)  
*Holds pointer to some user assigned data.*
- [HorizontalAlignment d\\_horzAlign](#)  
*Specifies the base for horizontal alignment.*
- [VerticalAlignment d\\_vertAlign](#)  
*Specifies the base for vertical alignment.*
- [UVector2 d\\_minSize](#)  
*current minimum size for the window.*
- [UVector2 d\\_maxSize](#)  
*current maximum size for the window.*
- [bool d\\_enabled](#)  
*true when [Window](#) is enabled*
- [bool d\\_visible](#)  
*true when [Window](#) is visible (that is it will be rendered, but may be obscured so no necessarily really visible)*
- [bool d\\_active](#)  
*true when [Window](#) is the active [Window](#) (receiving inputs).*
- [bool d\\_clippedByParent](#)  
*true when [Window](#) will be clipped by parent [Window](#) area [Rect](#).*
- [bool d\\_destroyedByParent](#)

*true when [Window](#) will be auto-destroyed by parent.*

- [bool d\\_alwaysOnTop](#)  
*true if [Window](#) will be drawn on top of all other Windows*
- [bool d\\_inheritsAlpha](#)  
*true if the [Window](#) inherits alpha from the parent [Window](#)*
- [bool d\\_restoreOldCapture](#)  
*true if the [Window](#) restores capture to the previous window when it releases capture.*
- [bool d\\_zOrderingEnabled](#)  
*true if the [Window](#) responds to z-order change requests.*
- [bool d\\_wantsMultiClicks](#)  
*true if the [Window](#) wishes to hear about multi-click mouse events.*
- [bool d\\_distCapturedInputs](#)  
*true if unhandled captured inputs should be distributed to child windows.*
- [bool d\\_riseOnClick](#)  
*True if the window should come to the front of the z order in response to a left mouse button down event.*
- [bool d\\_autoRepeat](#)  
*true if button will auto-repeat mouse button down events while mouse button is held down.*
- [float d\\_repeatDelay](#)  
*seconds before first repeat event is fired*
- [float d\\_repeatRate](#)  
*secons between further repeats after delay has expired.*
- [bool d\\_repeating](#)  
*implements repeating - is true after delay has elapsed,*
- [float d\\_repeatElapsed](#)  
*implements repeating - tracks time elapsed.*
- [MouseButton d\\_repeatButton](#)  
*Button we're tracking (implication of this is that we only support one button at a time).*
- [bool d\\_dragDropTarget](#)  
*true if window will receive drag and drop related notifications*
- [String d\\_tooltipText](#)  
*Text string used as tip for this window.*
- [Tooltip \\* d\\_customTip](#)  
*Possible custom [Tooltip](#) for this window.*

- bool [d\\_weOwnTip](#)  
*true if this [Window](#) created the custom [Tooltip](#).*
- bool [d\\_inheritsTipText](#)  
*true if the [Window](#) inherits tooltip text from its parent (when none set for itself).*
- [RenderCache](#) [d\\_renderCache](#)  
*Object which acts as a cache for Images to be drawn by this [Window](#).*
- bool [d\\_needsRedraw](#)  
*true if window image cache needs to be regenerated.*
- [String](#) [d\\_lookName](#)  
*Name of the Look assigned to this window (if any).*
- [WindowRenderer](#) \* [d\\_windowRenderer](#)  
*The [WindowRenderer](#) module that implements the Look'N'Feel specification.*
- bool [d\\_initialising](#)  
*true when this window is currently being initialised (creating children etc)*
- bool [d\\_destructionStarted](#)  
*true when this window is being destroyed.*
- bool [d\\_mousePassThroughEnabled](#)  
*true if this window can never be "hit" by the cursor. false for normal mouse event handling.*
- bool [d\\_autoWindow](#)  
*true when this window is an auto-window (it's name contains \_\_auto\_)*
- [BannedXMLPropertySet](#) [d\\_bannedXMLProperties](#)
- bool [d\\_allowWriteXML](#)  
*true if this window is allowed to write XML, false if not*
- [Rect](#) [d\\_screenUnclippedRect](#)  
*current unclipped screen rect in pixels*
- bool [d\\_screenUnclippedRectValid](#)
- [Rect](#) [d\\_screenUnclippedInnerRect](#)  
*current unclipped inner screen rect in pixels*
- bool [d\\_screenUnclippedInnerRectValid](#)
- [Rect](#) [d\\_screenRect](#)  
*current fully clipped screen rect in pixels*
- bool [d\\_screenRectValid](#)
- [Rect](#) [d\\_screenInnerRect](#)  
*current fully clipped inner screen rect in pixels*
- bool [d\\_screenInnerRectValid](#)

- `const String d_type`  
*String holding the type name for the [Window](#) (is also the name of the [WindowFactory](#) that created us).*
- `String d_name`  
*The name of the window (GUI system unique).*
- `String d_falagardType`  
*Type name of the window as defined in a Falagard mapping.*
- `String d_windowPrefix`  
*The prefix used on this window (if any) when created instanced windows.*

## Static Protected Attributes

- `static Window * d_captureWindow = 0`  
*Window that has captured inputs.*
- `static WindowProperties::Alpha d_alphaProperty`
- `static WindowProperties::AlwaysOnTop d_alwaysOnTopProperty`
- `static WindowProperties::ClippedByParent d_clippedByParentProperty`
- `static WindowProperties::DestroyedByParent d_destroyedByParentProperty`
- `static WindowProperties::Disabled d_disabledProperty`
- `static WindowProperties::Font d_fontProperty`
- `static WindowProperties::ID d_IDProperty`
- `static WindowProperties::InheritsAlpha d_inheritsAlphaProperty`
- `static WindowProperties::MouseCursorImage d_mouseCursorProperty`
- `static WindowProperties::RestoreOldCapture d_restoreOldCaptureProperty`
- `static WindowProperties::Text d_textProperty`
- `static WindowProperties::Visible d_visibleProperty`
- `static WindowProperties::ZOrderChangeEnabled d_zOrderChangeProperty`
- `static WindowProperties::WantsMultiClickEvents d_wantsMultiClicksProperty`
- `static WindowProperties::MouseButtonDownAutoRepeat d_autoRepeatProperty`
- `static WindowProperties::AutoRepeatDelay d_autoRepeatDelayProperty`
- `static WindowProperties::AutoRepeatRate d_autoRepeatRateProperty`
- `static WindowProperties::DistributeCapturedInputs d_distInputsProperty`
- `static WindowProperties::CustomTooltipType d_tooltipTypeProperty`
- `static WindowProperties::Tooltip d_tooltipProperty`
- `static WindowProperties::InheritsTooltipText d_inheritsTooltipProperty`
- `static WindowProperties::RiseOnClick d_riseOnClickProperty`
- `static WindowProperties::VerticalAlignment d_vertAlignProperty`
- `static WindowProperties::HorizontalAlignment d_horzAlignProperty`
- `static WindowProperties::UnifiedAreaRect d_unifiedAreaRectProperty`
- `static WindowProperties::UnifiedPosition d_unifiedPositionProperty`
- `static WindowProperties::UnifiedXPosition d_unifiedXPositionProperty`
- `static WindowProperties::UnifiedYPosition d_unifiedYPositionProperty`
- `static WindowProperties::UnifiedSize d_unifiedSizeProperty`
- `static WindowProperties::UnifiedWidth d_unifiedWidthProperty`
- `static WindowProperties::UnifiedHeight d_unifiedHeightProperty`
- `static WindowProperties::UnifiedMinSize d_unifiedMinSizeProperty`

- static [WindowProperties::UnifiedMaxSize](#) **d\_unifiedMaxSizeProperty**
- static [WindowProperties::MousePassThroughEnabled](#) **d\_mousePassThroughEnabledProperty**
- static [WindowProperties::WindowRenderer](#) **d\_windowRendererProperty**
- static [WindowProperties::LookNFeel](#) **d\_lookNFeelProperty**
- static [WindowProperties::DragDropTarget](#) **d\_dragDropTargetProperty**

## Friends

- class **System**
- class **WindowManager**

### 6.340.1 Detailed Description

An abstract base class providing common functionality and specifying the required interface for derived classes.

The [Window](#) base class is core UI object class that the the system knows about; for this reason, every other window, widget, or similar item within the system must be derived from [Window](#).

The base class provides the common functionality required by all UI objects, and specifies the minimal interface required to be implemented by derived classes.

### 6.340.2 Constructor & Destructor Documentation

#### 6.340.2.1 CEGUI::Window::Window (const String & *type*, const String & *name*)

Constructor for [Window](#) base class.

##### Parameters:

- type* [String](#) object holding [Window](#) type (usually provided by [WindowFactory](#)).
- name* [String](#) object holding unique name for the [Window](#).

### 6.340.3 Member Function Documentation

#### 6.340.3.1 const String & CEGUI::Window::getType (void) const

return a [String](#) object holding the type name for this [Window](#).

##### Returns:

[String](#) object holding the [Window](#) type.

#### 6.340.3.2 const String& CEGUI::Window::getName (void) const [inline]

return a [String](#) object holding the name of this [Window](#).

##### Returns:

[String](#) object holding the unique [Window](#) name.

**6.340.3.3** `const String& CEGUI::Window::getPrefix (void) const` `[inline]`

Return a string to the window prefix

**Returns:**

[String](#) object holding the prefix of this window

**6.340.3.4** `bool CEGUI::Window::isDestroyedByParent (void) const` `[inline]`

returns whether or not this [Window](#) is set to be destroyed when its parent window is destroyed.

**Returns:**

- true if the [Window](#) will be destroyed when its parent is destroyed.
- false if the [Window](#) will remain when its parent is destroyed.

**6.340.3.5** `bool CEGUI::Window::isAlwaysOnTop (void) const` `[inline]`

returns whether or not this [Window](#) is an always on top [Window](#). Also known as a top-most window.

**Returns:**

- true if this [Window](#) is always drawn on top of other normal windows.
- false if the [Window](#) has normal z-order behaviour.

**6.340.3.6** `bool CEGUI::Window::isDisabled (bool localOnly = false) const`

return whether the [Window](#) is currently disabled

**Parameters:**

*localOnly* States whether to only return the state set for this window, and not to factor in inherited state from ancestor windows.

**Returns:**

- true if the window is disabled.
- false if the window is enabled.

**6.340.3.7** `bool CEGUI::Window::isVisible (bool localOnly = false) const`

return true if the [Window](#) is currently visible.

When true is returned from this function does not mean that the window is not completely obscured by other windows, just that the window will be processed when rendering, and is not explicitly marked as hidden.

**Parameters:**

*localOnly* States whether to only return the state set for this window, and not to factor in inherited state from ancestor windows.



**Returns:**

- true if the window will be drawn.
- false if the window is hidden and therefore ignored when rendering.

**6.340.3.8 bool CEGUI::Window::isActive (void) const**

return true if this is the active [Window](#). An active window is a window that may receive user inputs.

Mouse events are always sent to the window containing the mouse cursor regardless of what this function reports (unless a window has captured inputs). The active state mainly determines where send other, for example keyboard, inputs.

**Returns:**

- true if the window is active and may be sent inputs by the system.
- false if the window is inactive and will not be sent inputs.

**6.340.3.9 bool CEGUI::Window::isClippedByParent (void) const [inline]**

return true if this [Window](#) is clipped so that its rendering will not pass outside of its parent [Window](#) area.

**Returns:**

- true if the window will be clipped by its parent [Window](#).
- false if the windows rendering may pass outside its parents area

**6.340.3.10 uint CEGUI::Window::getID (void) const [inline]**

return the ID code currently assigned to this [Window](#) by client code.

**Returns:**

uint value equal to the currently assigned ID code for this [Window](#).

**6.340.3.11 size\_t CEGUI::Window::getChildCount (void) const [inline]**

return the number of child [Window](#) objects currently attached to this [Window](#).

**Returns:**

size\_t value equal to the number of [Window](#) objects directly attached to this [Window](#) as children.

**6.340.3.12 bool CEGUI::Window::isChild (const String & name) const**

returns whether a [Window](#) with the specified name is currently attached to this [Window](#) as a child.

**Parameters:**

*name* [String](#) object containing the name of the [Window](#) to look for.

**Returns:**

- true if a [Window](#) named *name* is currently attached to this [Window](#).
- false if no such child [Window](#) is attached.

**6.340.3.13 bool CEGUI::Window::isChild (uint *ID*) const**

returns whether at least one window with the given ID code is attached to this [Window](#) as a child.

**Note:**

ID codes are client assigned and may or may not be unique, and as such, the return from this function will only have meaning to the client code.

**Parameters:**

*ID* uint ID code to look for.

**Returns:**

- true if at least one child window was found with the ID code *ID*
- false if no child window was found with the ID code *ID*.

**6.340.3.14 bool CEGUI::Window::isChildRecursive (uint *ID*) const**

returns whether at least one window with the given ID code is attached to this [Window](#) or any of it's children as a child.

**Note:**

ID codes are client assigned and may or may not be unique, and as such, the return from this function will only have meaning to the client code.

WARNING! This function can be very expensive and should only be used when you have no other option available. If you decide to use it anyway, make sure the window hierarchy from the entry point is small.

**Parameters:**

*ID* uint ID code to look for.

**Returns:**

- true if at least one child window was found with the ID code *ID*
- false if no child window was found with the ID code *ID*.

**6.340.3.15 bool CEGUI::Window::isChild (const Window \* *window*) const**

return true if the given [Window](#) is a child of this window.

**Parameters:**

*window* Pointer to the [Window](#) object to look for.

**Returns:**

- true if [Window](#) object *window* is attached to this window as a child.
- false if [Window](#) object *window* is not a child of this [Window](#).

**6.340.3.16 Window \* CEGUI::Window::getChild (const String & name) const**

return a pointer to the child window with the specified name.

This function will throw an exception if no child object with the given name is attached. This decision was made (over returning NULL if no window was found) so that client code can assume that if the call returns it has a valid window pointer. We provide the [isChild\(\)](#) functions for checking if a given window is attached.

**Parameters:**

*name* [String](#) object holding the name of the child window for which a pointer is to be returned.

**Returns:**

Pointer to the [Window](#) object attached to this window that has the name *name*.

**Exceptions:**

[UnknownObjectException](#) thrown if no window named *name* is attached to this [Window](#).

**6.340.3.17 Window \* CEGUI::Window::getChild (uint ID) const**

return a pointer to the first attached child window with the specified ID value.

This function will throw an exception if no child object with the given ID is attached. This decision was made (over returning NULL if no window was found) so that client code can assume that if the call returns it has a valid window pointer. We provide the [isChild\(\)](#) functions for checking if a given window is attached.

**Parameters:**

*ID* uint value specifying the ID code of the window to return a pointer to.

**Returns:**

Pointer to the (first) [Window](#) object attached to this window that has the ID code *ID*.

**Exceptions:**

[UnknownObjectException](#) thrown if no window with the ID code *ID* is attached to this [Window](#).

**6.340.3.18 Window \* CEGUI::Window::getChildRecursive (uint ID) const**

return a pointer to the first attached child window with the specified ID value. Children are traversed recursively.

Contrary to the non recursive version of this function, this one will not throw an exception, but return 0 in case no child was found.

**Note:**

WARNING! This function can be very expensive and should only be used when you have no other option available. If you decide to use it anyway, make sure the window hierarchy from the entry point is small.

**Parameters:**

*ID* uint value specifying the ID code of the window to return a pointer to.

**Returns:**

Pointer to the (first) [Window](#) object attached to this window that has the ID code *ID*. If no child is found with the ID code *ID*, 0 is returned.

**6.340.3.19 Window\* CEGUI::Window::getChildAtIdx (size\_t *idx*) const** `[inline]`

return a pointer to the child window that is attached to 'this' at the given index.

**Parameters:**

*idx* Index of the child window whos pointer should be returned. This value is not bounds checked, client code should ensure that this is less than the value returned by [getChildCount\(\)](#).

**Returns:**

Pointer to the child window currently attached at index position *idx*

**6.340.3.20 Window \* CEGUI::Window::getActiveChild (void)**

return a pointer to the [Window](#) that currently has input focus starting with this [Window](#).

**Returns:**

Pointer to the window that is active (has input focus) starting at this window. The function will return 'this' if this [Window](#) is active and either no children are attached or if none of the attached children are active. Returns NULL if this [Window](#) (and therefore all children) are not active.

**6.340.3.21 bool CEGUI::Window::isAncestor (const String & *name*) const**

return true if the specified [Window](#) is some ancestor of this [Window](#)

**Parameters:**

*name* [String](#) object holding the name of the [Window](#) to check for.

**Returns:**

- true if a [Window](#) named *name* is an ancestor (parent, or parent of parent, etc) of this [Window](#).
- false if a [Window](#) named *name* is in no way an ancestor of this window.

**6.340.3.22 bool CEGUI::Window::isAncestor (uint *ID*) const**

return true if any [Window](#) with the given ID is some ancestor of this [Window](#).

**Parameters:**

*ID* uint value specifying the ID to look for.

**Returns:**

- true if an ancestor (parent, or parent of parent, etc) was found with the ID code *ID*.
- false if no ancestor window has the ID code *ID*.

**6.340.3.23 bool CEGUI::Window::isAncestor (const Window \* *window*) const**

return true if the specified [Window](#) is some ancestor of this [Window](#).

**Parameters:**

*window* Pointer to the [Window](#) object to look for.

**Returns:**

- true if *window* was found to be an ancestor (parent, or parent of parent, etc) of this [Window](#).
- false if *window* is not an ancestor of this window.

**6.340.3.24 Font \* CEGUI::Window::getFont (bool *useDefault* = true) const**

return the active [Font](#) object for the [Window](#).

**Parameters:**

*useDefault* Specifies whether to return the default font if this [Window](#) has no preferred font set.

**Returns:**

Pointer to the [Font](#) being used by this [Window](#). If the window has no assigned font, and *useDefault* is true, then the default system font is returned.

**6.340.3.25 const String& CEGUI::Window::getText (void) const [inline]**

return the current text for the [Window](#)

**Returns:**

The [String](#) object that holds the current text for this [Window](#).

**6.340.3.26 bool CEGUI::Window::inheritsAlpha (void) const** [inline]

return true if the [Window](#) inherits alpha from its parent(s).

**Returns:**

- true if the [Window](#) inherits alpha from its parent(s)
- false if the alpha for this [Window](#) is independant from its parents.

**6.340.3.27 float CEGUI::Window::getAlpha (void) const** [inline]

return the current alpha value set for this [Window](#)

**Note:**

The alpha value set for any given window may or may not be the final alpha value that is used when rendering. All window objects, by default, inherit alpha from thier parent window(s) - this will blend child windows, relatively, down the line of inheritance. This behaviour can be overridden via the [setInheritsAlpha\(\)](#) method. To return the true alpha value that will be applied when rendering, use the [getEffectiveAlpha\(\)](#) method.

**Returns:**

the currently set alpha value for this [Window](#). The value returned Will be between 0.0f and 1.0f.

**6.340.3.28 float CEGUI::Window::getEffectiveAlpha (void) const**

return the effective alpha value that will be used when rendering this window, taking into account inheritance of parent window(s) alpha.

**Returns:**

the effective alpha that will be applied to this [Window](#) when rendering. The value returned Will be between 0.0f and 1.0f.

**6.340.3.29 Rect CEGUI::Window::getPixelRect (void) const**

return a [Rect](#) object describing the [Window](#) area in screen space.

**Returns:**

[Rect](#) object that describes the area covered by the [Window](#). The values in the returned [Rect](#) are in screen pixels. The returned [Rect](#) is clipped as appropriate and depending upon the 'ClippedByParent' setting.

**Note:**

This has now been made virtual to ease some customisations that require more specialised clipping requirements.

**6.340.3.30 Rect CEGUI::Window::getPixelRect\_impl (void) const** [virtual]

return a [Rect](#) object describing the [Window](#) area in screen space.

**Returns:**

[Rect](#) object that describes the area covered by the [Window](#). The values in the returned [Rect](#) are in screen pixels. The returned [Rect](#) is clipped as appropriate and depending upon the 'ClippedByParent' setting.

**Note:**

This has now been made virtual to ease some customisations that require more specialised clipping requirements.

**6.340.3.31 Rect CEGUI::Window::getInnerRect (void) const**

return a [Rect](#) object describing the clipped inner area for this window.

**Returns:**

[Rect](#) object that describes, in appropriately clipped screen pixel co-ordinates, the window object's inner rect area.

**6.340.3.32 Rect CEGUI::Window::getUnclippedPixelRect (void) const**

return a [Rect](#) object describing the [Window](#) area unclipped, in screen space.

**Returns:**

[Rect](#) object that describes the area covered by the [Window](#). The values in the returned [Rect](#) are in screen pixels. The returned rect is fully unclipped.

**6.340.3.33 Rect CEGUI::Window::getUnclippedInnerRect (void) const**

Return a [Rect](#) object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.

**Returns:**

[Rect](#) object that describes, in unclipped screen pixel co-ordinates, the window object's inner rect area.

**6.340.3.34 Rect CEGUI::Window::getUnclippedInnerRect\_impl (void) const** [virtual]

Return a [Rect](#) object that describes, unclipped, the inner rectangle for this window. The inner rectangle is typically an area that excludes some frame or other rendering that should not be touched by subsequent rendering.

**Returns:**

[Rect](#) object that describes, in unclipped screen pixel co-ordinates, the window object's inner rect area.

Reimplemented in [CEGUI::ClippedContainer](#), and [CEGUI::ScrolledContainer](#).

**6.340.3.35 static Window\* CEGUI::Window::getCaptureWindow (void) [inline, static]**

return the [Window](#) that currently has inputs captured.

**Returns:**

Pointer to the [Window](#) object that currently has inputs captured, or NULL if no [Window](#) has captured input.

**6.340.3.36 bool CEGUI::Window::isCapturedByThis (void) const [inline]**

return true if this [Window](#) has input captured.

**Returns:**

- true if this [Window](#) has captured inputs.
- false if some other [Window](#), or no [Window](#), has captured inputs.

**6.340.3.37 bool CEGUI::Window::isCapturedByAncestor (void) const [inline]**

return true if an ancestor window has captured inputs.

**Returns:**

- true if input is captured by a [Window](#) that is some ancestor (parent, parent of parent, etc) of this [Window](#).
- false if no ancestor of this window has captured input.

**6.340.3.38 bool CEGUI::Window::isCapturedByChild (void) const [inline]**

return true if a child window has captured inputs.

**Returns:**

- true if input is captured by a [Window](#) that is a child of this [Window](#).
- false if no child of this window has not captured input.

**6.340.3.39 bool CEGUI::Window::isHit (const Vector2 & *position*) const [virtual]**

check if the given pixel position would hit this window.

**Parameters:**

*position* [Vector2](#) object describing the position to check. The position describes a pixel offset from the top-left corner of the display.

**Returns:**

- true if *position* hits this [Window](#).
- false if *position* does not hit this window.

Reimplemented in [CEGUI::Combobox](#), and [CEGUI::FrameWindow](#).



**6.340.3.40 Window \* CEGUI::Window::getChildAtPosition (const Vector2 & *position*) const**

return the child [Window](#) that is hit by the given pixel position

**Parameters:**

*position* [Vector2](#) object describing the position to check. The position describes a pixel offset from the top-left corner of the display.

**Returns:**

Pointer to the child [Window](#) that was hit according to the location *position*, or 0 if no child of this window was hit.

**6.340.3.41 Window \* CEGUI::Window::getTargetChildAtPosition (const Vector2 & *position*) const**

return the child [Window](#) that is 'hit' by the given position, and is allowed to handle mouse events.

**Parameters:**

*position* [Vector2](#) object describing the position to check. The position describes a pixel offset from the top-left corner of the display.

**Returns:**

Pointer to the child [Window](#) that was hit according to the location *position*, or 0 if no child of this window was hit.

**6.340.3.42 Window\* CEGUI::Window::getParent (void) const [inline]**

return the parent of this [Window](#).

**Returns:**

Pointer to the [Window](#) object that is the parent of this [Window](#). This value can be NULL, in which case the [Window](#) is a GUI sheet / root.

**6.340.3.43 const Image \* CEGUI::Window::getMouseCursor (bool *useDefault* = true) const**

Return a pointer to the mouse cursor image to use when the mouse cursor is within this window's area.

**Parameters:**

*useDefault* Specifies whether to return the default mouse cursor image if this window specifies no preferred mouse cursor image.

**Returns:**

Pointer to the mouse cursor image that will be used when the mouse enters this window's area. May return NULL indicating no cursor will be drawn for this window.

**6.340.3.44** `Size CEGUI::Window::getPixelSize (void) const` `[inline]`

Return the window size in pixels.

**Returns:**

[Size](#) object describing this windows size in pixels.

**6.340.3.45** `void* CEGUI::Window::getUserData (void) const` `[inline]`

Return the user data set for this [Window](#).

Each [Window](#) can have some client assigned data attached to it, this data is not used by the GUI system in any way. Interpretation of the data is entirely application specific.

**Returns:**

pointer to the user data that is currently set for this window.

**6.340.3.46** `bool CEGUI::Window::restoresOldCapture (void) const` `[inline]`

Return whether this window is set to restore old input capture when it loses input capture.

This is only really useful for certain sub-components for widget writers.

**Returns:**

- true if the window will restore the previous capture window when it loses input capture.
- false if the window will set the capture window to NULL when it loses input capture (this is the default behaviour).

**6.340.3.47** `bool CEGUI::Window::isZOrderingEnabled (void) const`

Return whether z-order changes are enabled or disabled for this [Window](#).

**Returns:**

- true if z-order changes are enabled for this window. `moveToFront/moveToBack` work normally as expected.
- false: z-order changes are disabled for this window. `moveToFront/moveToBack` are ignored for this window.

**6.340.3.48** `bool CEGUI::Window::wantsMultiClickEvents (void) const`

Return whether this window will receive multi-click events or multiple 'down' events instead.

**Returns:**

- true if the [Window](#) will receive double-click and triple-click events.
- false if the [Window](#) will receive multiple mouse button down events instead of double/triple click events.

**6.340.3.49 bool CEGUI::Window::isMouseAutoRepeatEnabled (void) const**

Return whether mouse button down event autorepeat is enabled for this window.

**Returns:**

- true if autorepeat of mouse button down events is enabled for this window.
- false if autorepeat of mouse button down events is not enabled for this window.

**6.340.3.50 float CEGUI::Window::getAutoRepeatDelay (void) const**

Return the current auto-repeat delay setting for this window.

**Returns:**

float value indicating the delay, in seconds, before the first repeat mouse button down event will be triggered when autorepeat is enabled.

**6.340.3.51 float CEGUI::Window::getAutoRepeatRate (void) const**

Return the current auto-repeat rate setting for this window.

**Returns:**

float value indicating the rate, in seconds, at which repeat mouse button down events will be generated after the initial delay has expired.

**6.340.3.52 bool CEGUI::Window::distributesCapturedInputs (void) const**

Return whether the window wants inputs passed to its attached child windows when the window has inputs captured.

**Returns:**

- true if [System](#) should pass captured input events to child windows.
- false if [System](#) should pass captured input events to this window only.

**6.340.3.53 bool CEGUI::Window::isUsingDefaultTooltip (void) const**

Return whether this [Window](#) is using the system default [Tooltip](#) for its [Tooltip](#) window.

**Returns:**

- true if the [Window](#) will use the system default tooltip.
- false if the window has a custom [Tooltip](#) object.

**6.340.3.54   `Tooltip *` `CEGUI::Window::getTooltip (void) const`**

Return a pointer to the [Tooltip](#) object used by this [Window](#). The value returned may point to the system default [Tooltip](#), a custom [Window](#) specific [Tooltip](#), or be NULL.

**Returns:**

Pointer to a [Tooltip](#) based object, or NULL.

**6.340.3.55   `String` `CEGUI::Window::getTooltipType (void) const`**

Return the custom tooltip type.

**Returns:**

[String](#) object holding the current custom tooltip window type, or an empty string if no custom tooltip is set.

**6.340.3.56   `const String &` `CEGUI::Window::getTooltipText (void) const`**

Return the current tooltip text set for this [Window](#).

**Returns:**

[String](#) object holding the current tooltip text set for this window.

**6.340.3.57   `bool` `CEGUI::Window::inheritsTooltipText (void) const`**

Return whether this window inherits [Tooltip](#) text from its parent when its own tooltip text is not set.

**Returns:**

- true if the window inherits tooltip text from its parent when its own text is not set.
- false if the window does not inherit tooltip text from its parent (and shows no tooltip when no text is set).

**6.340.3.58   `bool` `CEGUI::Window::isRiseOnClickEnabled (void) const`   `[inline]`**

Return whether this window will rise to the top of the z-order when clicked with the left mouse button.

**Returns:**

- true if the window will come to the top of other windows when the left mouse button is pushed within its area.
- false if the window does not change z-order position when the left mouse button is pushed within its area.

**6.340.3.59** `bool CEGUI::Window::testClassName (const String & class_name) const` `[inline]`

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

**Parameters:**

*class\_name* The class name that is to be checked.

**Returns:**

true if this window was inherited from *class\_name*. false if not.

**6.340.3.60** `VerticalAlignment CEGUI::Window::getVerticalAlignment () const` `[inline]`

Get the vertical alignment.

Returns the vertical alignment for the window. This setting affects how the windows position is interpreted relative to its parent.

**Returns:**

One of the VerticalAlignment enumerated values.

**6.340.3.61** `HorizontalAlignment CEGUI::Window::getHorizontalAlignment () const` `[inline]`

Get the horizontal alignment.

Returns the horizontal alignment for the window. This setting affects how the windows position is interpreted relative to its parent.

**Returns:**

One of the HorizontalAlignment enumerated values.

**6.340.3.62** `RenderCache& CEGUI::Window::getRenderCache ()` `[inline]`

Return the [RenderCache](#) object for this [Window](#).

**Returns:**

Reference to the [RenderCache](#) object for this [Window](#).

**6.340.3.63** `const String & CEGUI::Window::getLookNFeel () const`

Get the name of the LookNFeel assigned to this window.

**Returns:**

[String](#) object holding the name of the look assigned to this window. Returns the empty string if no look is assigned.

**6.340.3.64** `bool CEGUI::Window::getModalState (void) const` `[inline]`

Get whether or not this [Window](#) is the modal target.

**Returns:**

Returns true if this [Window](#) is the modal target, otherwise false.

**6.340.3.65** `const String & CEGUI::Window::getUserString (const String & name) const`

Returns a named user string.

**Parameters:**

*name* [String](#) object holding the name of the string to be returned.

**Returns:**

[String](#) object holding the data stored for the requested user string.

**Exceptions:**

[UnknownObjectException](#) thrown if a user string named *name* does not exist.

**6.340.3.66** `bool CEGUI::Window::isUserStringDefined (const String & name) const`

Return whether a user string with the specified name exists.

**Parameters:**

*name* [String](#) object holding the name of the string to be checked.

**Returns:**

- true if a user string named *name* exists.
- false if no such user string exists.

**6.340.3.67** `Window * CEGUI::Window::getActiveSibling ()`

Returns the active sibling window.

This searches the immediate children of this window's parent, and returns a pointer to the active window. The method will return this if we are the immediate child of our parent that is active. If our parent is not active, or if no immediate child of our parent is active then 0 is returned. If this window has no parent, and this window is not active then 0 is returned, else this is returned.

**Returns:**

A pointer to the immediate child window attached to our parent that is currently active, or 0 if no immediate child of our parent is active.

**6.340.3.68 Size CEGUI::Window::getParentPixelSize (void) const**

Return the pixel size of the parent element. This always returns a valid object.

**Returns:**

[Size](#) object that describes the pixel dimensions of this [Window](#) objects parent

**6.340.3.69 float CEGUI::Window::getParentPixelWidth (void) const**

Return the pixel Width of the parent element. This always returns a valid number.

**Returns:**

float value that is equal to the pixel width of this [Window](#) objects parent

**6.340.3.70 float CEGUI::Window::getParentPixelHeight (void) const**

Return the pixel Height of the parent element. This always returns a valid number.

**Returns:**

float value that is equal to the pixel height of this [Window](#) objects parent

**6.340.3.71 bool CEGUI::Window::isMousePassThroughEnabled (void) const [inline]**

Returns whether this window should ignore mouse event and pass them through to and other windows behind it. In effect making the window transparent to the mouse.

**Returns:**

true if mouse pass through is enabled. false if mouse pass through is not enabled.

**6.340.3.72 EventSet::Iterator CEGUI::Window::getEventIterator () const**

Helper method that returns an EventSet::Iterator object that can be used to iterate over the events currently added to the [EventSet](#) of this [Window](#).

This helper member is provided as an easy way to avoid some abiguity we have due to using multiple inheritance. Ultimately it avoids the need to do things like this (which some people don't like!):

```
obtain an iterator for the EventSet
EventSet::Iterator evt_iter = myWindow->EventSet::getIterator();

obtain an iterator for the PropertySet
PropertySet::Iterator prp_iter = myWindow->PropertySet::getIterator();
```

**Note:**

Iterating over events in the [EventSet](#) is of questionable use these days, since available Events are no longer added in one batch at creation time, but are added individually whenever an event is first subscribed.

**Returns:**

EventSet::Iterator object.

**6.340.3.73 PropertySet::Iterator CEGUI::Window::getPropertyIterator () const**

Helper method that returns a PropertySet::Iterator object that can be used to iterate over the events currently added to the [PropertySet](#) of this [Window](#).

This helper member is provided as an easy way to avoid some ambiguity we have due to using multiple inheritance. Ultimately it avoids the need to do things like this (which some people don't like!):

```
obtain an iterator for the EventSet
    EventSet::Iterator evt_iter = myWindow->EventSet::getIterator();

obtain an iterator for the PropertySet
    PropertySet::Iterator prp_iter = myWindow->PropertySet::getIterator();
```

**Returns:**

PropertySet::Iterator object.

**6.340.3.74 bool CEGUI::Window::isDragDropTarget () const**

Returns whether this [Window](#) object will receive events generated by the drag and drop support in the system.

**Returns:**

- true if the [Window](#) is enabled as a drag and drop target.
- false if the window is not enabled as a drag and drop target.

**6.340.3.75 void CEGUI::Window::rename (const String & new\_name)**

Renames the window.

**Parameters:**

*new\_name* [String](#) object holding the new name for the window.

**Exceptions:**

[AlreadyExistsException](#) thrown if a [Window](#) named *new\_name* already exists in the system.

**6.340.3.76 virtual void CEGUI::Window::initialiseComponents (void) [inline, virtual]**

Initialises the [Window](#) based object ready for use.

**Note:**

This must be called for every window created. Normally this is handled automatically by the [Window-Manager](#).



**Returns:**

Nothing

Reimplemented in [CEGUI::Combobox](#), [CEGUI::ComboDropList](#), [CEGUI::FrameWindow](#), [CEGUI::GroupBox](#), [CEGUI::ItemListBase](#), [CEGUI::Listbox](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::ScrollablePane](#), [CEGUI::Scrollbar](#), [CEGUI::ScrolledItemListBase](#), [CEGUI::Slider](#), [CEGUI::Spinner](#), and [CEGUI::TabControl](#).

**6.340.3.77 void CEGUI::Window::setDestroyedByParent (bool *setting*)**

Set whether or not this [Window](#) will automatically be destroyed when its parent [Window](#) is destroyed.

**Parameters:**

- setting* • true to have the [Window](#) auto-destroyed when its parent is destroyed (default behaviour)
- false to have the [Window](#) remain after its parent is destroyed.

**Returns:**

Nothing

**6.340.3.78 void CEGUI::Window::setAlwaysOnTop (bool *setting*)**

Set whether this window is always on top, or not.

**Parameters:**

- setting* • true to have the [Window](#) appear on top of all other non always on top windows
- false to allow the window to be covered by other normal windows.

**Returns:**

Nothing

**6.340.3.79 void CEGUI::Window::setEnabled (bool *setting*)**

Set whether this window is enabled or disabled. A disabled window normally can not be interacted with, and may have different rendering.

**Parameters:**

- setting* • true to enable the [Window](#)
- false to disable the [Window](#).

**Returns:**

Nothing

**6.340.3.80 void CEGUI::Window::enable (void) [inline]**

enable the [Window](#) to allow interaction.

**Returns:**

Nothing

**6.340.3.81 void CEGUI::Window::disable (void) [inline]**

disable the [Window](#) to prevent interaction.

**Returns:**

Nothing

**6.340.3.82 void CEGUI::Window::setVisible (bool *setting*)**

Set whether the [Window](#) is visible or hidden.

**Parameters:**

- setting* • true to make the [Window](#) visible.
- false to make the [Window](#) hidden.

**Returns:**

Nothing

**6.340.3.83 void CEGUI::Window::show (void) [inline]**

show the [Window](#)

**Returns:**

Nothing

**6.340.3.84 void CEGUI::Window::hide (void) [inline]**

hide the [Window](#).

**Returns:**

Nothing

**6.340.3.85 void CEGUI::Window::activate (void)**

Activate the [Window](#) giving it input focus and bringing it to the top of all windows with the same always-on-top settig as this [Window](#).

**Returns:**

Nothing

**6.340.3.86 void CEGUI::Window::deactivate (void)**

Deactivate the window. No further inputs will be received by the window until it is re-activated either programmatically or by the user interacting with the gui.

**Returns:**

Nothing.

**6.340.3.87 void CEGUI::Window::setClippedByParent (bool *setting*)**

Set whether this [Window](#) will be clipped by its parent window(s).

**Parameters:**

- setting*
- true to have the [Window](#) clipped so that rendering is constrained to within the area of its parent(s).
  - false to have rendering constrained to the screen only.

**Returns:**

Nothing

**6.340.3.88 void CEGUI::Window::setID (uint *ID*)**

Set the current ID for the [Window](#).

**Parameters:**

- ID* Client assigned ID code for this [Window](#). The GUI system assigns no meaning to any IDs, they are a device purely for client code usage.

**Returns:**

Nothing

**6.340.3.89 void CEGUI::Window::setPrefix (String *prefix*) [inline]**

Sets the unique prefix for this window.

**Parameters:**

- prefix* [String](#) object holding the prefix to be used on this window.

**6.340.3.90 void CEGUI::Window::setText (const String & *text*)**

Set the current text string for the [Window](#).

**Parameters:**

- text* [String](#) object containing the text that is to be set as the [Window](#) text.

**Returns:**

Nothing

**6.340.3.91 void CEGUI::Window::setFont (Font \* *font*)**

Set the font used by this [Window](#).

**Parameters:**

*font* Pointer to the [Font](#) object to be used by this [Window](#). If *font* is NULL, the default font will be used.

**Returns:**

Nothing

**6.340.3.92 void CEGUI::Window::setFont (const String & *name*)**

Set the font used by this [Window](#).

**Parameters:**

*name* [String](#) object holding the name of the [Font](#) object to be used by this [Window](#). If *name* == "", the default font will be used.

**Returns:**

Nothing

**Exceptions:**

[UnknownObjectException](#) thrown if the specified [Font](#) is unknown within the system.

**6.340.3.93 void CEGUI::Window::addChildWindow (const String & *name*)**

Add the named [Window](#) as a child of this [Window](#). If the [Window](#) *name* is already attached to a [Window](#), it is detached before being added to this [Window](#).

**Parameters:**

*name* [String](#) object holding the name of the [Window](#) to be added.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if no [Window](#) named *name* exists.

[InvalidRequestException](#) thrown if [Window](#) *name* is an ancestor of this [Window](#), to prevent cyclic [Window](#) structures.

**6.340.3.94 void CEGUI::Window::addChildWindow (Window \* *window*)**

Add the specified [Window](#) as a child of this [Window](#). If the [Window](#) *window* is already attached to a [Window](#), it is detached before being added to this [Window](#).

**Parameters:**

*window* Pointer to the [Window](#) object to be added.

**Returns:**

Nothing

**Exceptions:**

[InvalidRequestException](#) thrown if [Window](#) *window* is an ancestor of this [Window](#), to prevent cyclic [Window](#) structures.

**6.340.3.95 void CEGUI::Window::removeChildWindow (const String & *name*)**

Remove the named [Window](#) from this windows child list.

**Parameters:**

*name* [String](#) object holding the name of the [Window](#) to be removed. If the [Window](#) specified is not attached to this [Window](#), nothing happens.

**Returns:**

Nothing.

**6.340.3.96 void CEGUI::Window::removeChildWindow (Window \* *window*)**

Remove the specified [Window](#) form this windows child list.

**Parameters:**

*window* Pointer to the [Window](#) object to be removed. If the *window* is not attached to this [Window](#), then nothing happens.

**Returns:**

Nothing.

**6.340.3.97 void CEGUI::Window::removeChildWindow (uint *ID*)**

Remove the first child [Window](#) with the specified ID. If there is more than one attached [Window](#) objects with the specified ID, only the fist one encountered will be removed.

**Parameters:**

*ID* ID number assigned to the [Window](#) to be removed. If no [Window](#) with ID code *ID* is attached, nothing happens.

**Returns:**

Nothing.

**6.340.3.98 void CEGUI::Window::moveToFront ()**

Move the [Window](#) to the top of the z order.

- If the [Window](#) is a non always-on-top window it is moved the the top of all other non always-on-top sibling windows, and the process repeated for all ancestors.
- If the [Window](#) is an always-on-top window it is moved to the of of all sibling Windows, and the process repeated for all ancestors.

**Returns:**

Nothing

**6.340.3.99 void CEGUI::Window::moveToBack ()**

Move the [Window](#) to the bottom of the Z order.

- If the window is non always-on-top the [Window](#) is sent to the very bottom of its sibling windows and the process repeated for all ancestors.
- If the window is always-on-top, the [Window](#) is sent to the bottom of all sibling always-on-top windows and the process repeated for all ancestors.

**Returns:**

Nothing

**6.340.3.100 bool CEGUI::Window::captureInput (void)**

Captures input to this window.

**Returns:**

- true if input was successfully captured to this window.
- false if input could not be captured to this window (maybe because the window is not active).

**6.340.3.101 void CEGUI::Window::releaseInput (void)**

Releases input capture from this [Window](#). If this [Window](#) does not have inputs captured, nothing happens.

**Returns:**

Nothing

**6.340.3.102 void CEGUI::Window::setRestoreCapture (bool *setting*)**

Set whether this window will remember and restore the previous window that had inputs captured.

**Parameters:**

- setting*
  - true: The window will remember and restore the previous capture window. The CaptureLost event is not fired on the previous window when this window steals input capture. When this window releases capture, the old capture window is silently restored.
  - false: Input capture works as normal, each window losing capture is signalled via CaptureLost, and upon the final release of capture, no previous setting is restored (this is the default behaviour).

**Returns:**

Nothing

**6.340.3.103 void CEGUI::Window::setAlpha (float *alpha*)**

Set the current alpha value for this window.

**Note:**

The alpha value set for any given window may or may not be the final alpha value that is used when rendering. All window objects, by default, inherit alpha from their parent window(s) - this will blend child windows, relatively, down the line of inheritance. This behaviour can be overridden via the [setInheritsAlpha\(\)](#) method. To return the true alpha value that will be applied when rendering, use the [getEffectiveAlpha\(\)](#) method.

**Parameters:**

*alpha* The new alpha value for the window. Value should be between 0.0f and 1.0f.

**Returns:**

Nothing

**6.340.3.104 void CEGUI::Window::setInheritsAlpha (bool *setting*)**

Sets whether this [Window](#) will inherit alpha from its parent windows.

**Parameters:**

- setting*
  - true if the [Window](#) should use inherited alpha.
  - false if the [Window](#) should have an independant alpha value.

**Returns:**

Nothing

**6.340.3.105 void CEGUI::Window::requestRedraw (void) const**

Signal the [System](#) object to redraw (at least) this [Window](#) on the next render cycle.

**Returns:**

Nothing

**6.340.3.106 void CEGUI::Window::setMouseCursor (const Image \* *image*) [inline]**

Set the mouse cursor image to be used when the mouse enters this window.

**Parameters:**

*image* Pointer to the [Image](#) object to use as the mouse cursor image when the mouse enters the area for this [Window](#).

**Returns:**

Nothing.

**6.340.3.107 void CEGUI::Window::setMouseCursor (MouseCursorImage *image*) [inline]**

Set the mouse cursor image to be used when the mouse enters this window.

**Parameters:**

*image* One of the MouseCursorImage enumerated values.

**Returns:**

Nothing.

**6.340.3.108 void CEGUI::Window::setMouseCursor (const String & *imageset*, const String & *image\_name*)**

Set the mouse cursor image to be used when the mouse enters this window.

**Parameters:**

*imageset* [String](#) object that contains the name of the [Imageset](#) that contains the image to be used.

*image\_name* [String](#) object that contains the name of the [Image](#) on *imageset* that is to be used.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if *imageset* is not known, or if *imageset* contains no [Image](#) named *image\_name*.



**6.340.3.109 void CEGUI::Window::setUserData (void \* *user\_data*) [inline]**

Set the user data set for this [Window](#).

Each [Window](#) can have some client assigned data attached to it, this data is not used by the GUI system in any way. Interpretation of the data is entirely application specific.

**Parameters:**

*user\_data* pointer to the user data that is to be set for this window.

**Returns:**

Nothing.

**6.340.3.110 void CEGUI::Window::setZOrderingEnabled (bool *setting*)**

Set whether z-order changes are enabled or disabled for this [Window](#).

**Parameters:**

- setting*
- true if z-order changes are enabled for this window. `moveToFront/moveToBack` work normally as expected.
  - false: z-order changes are disabled for this window. `moveToFront/moveToBack` are ignored for this window.

**Returns:**

Nothing.

**6.340.3.111 void CEGUI::Window::setWantsMultiClickEvents (bool *setting*)**

Set whether this window will receive multi-click events or multiple 'down' events instead.

**Parameters:**

- setting*
- true if the [Window](#) will receive double-click and triple-click events.
  - false if the [Window](#) will receive multiple mouse button down events instead of double/triple click events.

**Returns:**

Nothing.

**6.340.3.112 void CEGUI::Window::setMouseAutoRepeatEnabled (bool *setting*)**

Set whether mouse button down event autorepeat is enabled for this window.

**Parameters:**

- setting*
- true to enable autorepeat of mouse button down events.
  - false to disable autorepeat of mouse button down events.

**Returns:**

Nothing.

**6.340.3.113 void CEGUI::Window::setAutoRepeatDelay (float *delay*)**

Set the current auto-repeat delay setting for this window.

**Parameters:**

*delay* float value indicating the delay, in seconds, before the first repeat mouse button down event should be triggered when autorepeat is enabled.

**Returns:**

Nothing.

**6.340.3.114 void CEGUI::Window::setAutoRepeatRate (float *rate*)**

Set the current auto-repeat rate setting for this window.

**Parameters:**

*rate* float value indicating the rate, in seconds, at which repeat mouse button down events should be generated after the initial delay has expired.

**Returns:**

Nothing.

**6.340.3.115 void CEGUI::Window::setDistributesCapturedInputs (bool *setting*)**

Set whether the window wants inputs passed to its attached child windows when the window has inputs captured.

**Parameters:**

- setting* • true if [System](#) should pass captured input events to child windows.
- false if [System](#) should pass captured input events to this window only.

**6.340.3.116 void CEGUI::Window::destroy (void) [virtual]**

Internal destroy method which actually just adds the window and any parent destructed child windows to the dead pool.

This is virtual to allow for specialised cleanup which may be required in some advanced cases. If you override this for the above reason, you MUST call this base class version.

**Note:**

You never have to call this method yourself, use [WindowManager](#) to destroy your [Window](#) objects (which will call this for you).

Reimplemented in [CEGUI::ScrollablePane](#).

**6.340.3.117 void CEGUI::Window::setTooltip (Tooltip \* *tooltip*)**

Set the custom [Tooltip](#) object for this [Window](#). This value may be 0 to indicate that the [Window](#) should use the system default [Tooltip](#) object.

**Parameters:**

*tooltip* Pointer to a valid [Tooltip](#) based object which should be used as the tooltip for this [Window](#), or 0 to indicate that the [Window](#) should use the system default [Tooltip](#) object. Note that when passing a pointer to a [Tooltip](#) object, ownership of the [Tooltip](#) does not pass to this [Window](#) object.

**Returns:**

Nothing.

**6.340.3.118 void CEGUI::Window::setTooltipType (const String & *tooltipType*)**

Set the custom [Tooltip](#) to be used by this [Window](#) by specifying a [Window](#) type.

The [Window](#) will internally attempt to create an instance of the specified window type (which must be derived from the base [Tooltip](#) class). If the [Tooltip](#) creation fails, the error is logged and the [Window](#) will revert to using either the existing custom [Tooltip](#) or the system default [Tooltip](#).

**Parameters:**

*tooltipType* [String](#) object holding the name of the [Tooltip](#) based [Window](#) type which should be used as the [Tooltip](#) for this [Window](#).

**Returns:**

Nothing.

**6.340.3.119 void CEGUI::Window::setTooltipText (const String & *tip*)**

Set the tooltip text for this window.

**Parameters:**

*tip* [String](#) object holding the text to be displayed in the tooltip for this [Window](#).

**Returns:**

Nothing.

**6.340.3.120 void CEGUI::Window::setInheritsTooltipText (bool *setting*)**

Set whether this window inherits [Tooltip](#) text from its parent when its own tooltip text is not set.

**Parameters:**

- setting*
- true if the window should inherit tooltip text from its parent when its own text is not set.
  - false if the window should not inherit tooltip text from its parent (and so show no tooltip when no text is set).

**Returns:**

Nothing.

**6.340.3.121 void CEGUI::Window::setRiseOnClickEnabled (bool *setting*) [inline]**

Set whether this window will rise to the top of the z-order when clicked with the left mouse button.

**Parameters:**

- setting* • true if the window should come to the top of other windows when the left mouse button is pushed within its area.
- false if the window should not change z-order position when the left mouse button is pushed within its area.

**Returns:**

Nothing.

**6.340.3.122 void CEGUI::Window::setVerticalAlignment (const VerticalAlignment *alignment*)**

Set the vertical alignment.

Modifies the vertical alignment for the window. This setting affects how the windows position is interpreted relative to its parent.

**Parameters:**

*alignment* One of the VerticalAlignment enumerated values.

**Returns:**

Nothing.

**6.340.3.123 void CEGUI::Window::setHorizontalAlignment (const HorizontalAlignment *alignment*)**

Set the horizontal alignment.

Modifies the horizontal alignment for the window. This setting affects how the windows position is interpreted relative to its parent.

**Parameters:**

*alignment* One of the HorizontalAlignment enumerated values.

**Returns:**

Nothing.

**6.340.3.124 void CEGUI::Window::setLookAndFeel (const String & *look*)** [virtual]

Set the LookAndFeel that should be used for this window.

**Parameters:**

*look* [String](#) object holding the name of the look to be assigned to the window.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if the look's feel specified by *look* does not exist.

**Note:**

Once a look's feel has been assigned it is locked - as it cannot be changed.

Reimplemented in [CEGUI::Tree](#).

**6.340.3.125 void CEGUI::Window::setModalState (bool *state*)**

Set the modal state for this [Window](#).

**Parameters:**

*state* Boolean value defining if this [Window](#) should be the modal target.

- true if this [Window](#) should be activated and set as the modal target.
- false if the modal target should be cleared if this [Window](#) is currently the modal target.

**Returns:**

Nothing.

**6.340.3.126 void CEGUI::Window::performChildWindowLayout (void)** [virtual]

method called to perform extended laying out of attached child windows.

The system may call this at various times (like when it is resized for example), and it may be invoked directly where required.

**Returns:**

Nothing.

Reimplemented in [CEGUI::ItemListBase](#), and [CEGUI::TabControl](#).

**6.340.3.127 void CEGUI::Window::setUserString (const String & *name*, const String & *value*)**

Sets the value of a named user string, creating it as required.

**Parameters:**

*name* [String](#) object holding the name of the string to be returned.

*value* [String](#) object holding the value to be assigned to the user string.

**Returns:**

Nothing.

**6.340.3.128 void CEGUI::Window::setArea (const UDim & *xpos*, const UDim & *ypos*, const UDim & *width*, const UDim & *height*)**

Set the window area.

Sets the area occupied by this window. The defined area is offset from the top-left corner of this windows parent window or from the top-left corner of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*xpos* [UDim](#) describing the new x co-ordinate (left edge) of the window area.

*ypos* [UDim](#) describing the new y co-ordinate (top-edge) of the window area.

*width* [UDim](#) describing the new width of the window area.

*height* [UDim](#) describing the new height of the window area.

**6.340.3.129 void CEGUI::Window::setArea (const UVector2 & *pos*, const UVector2 & *size*)**

Set the window area.

Sets the area occupied by this window. The defined area is offset from the top-left corner of this windows parent window or from the top-left corner of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*pos* [UVector2](#) describing the new position (top-left corner) of the window area.

*size* [UVector2](#) describing the new size of the window area.

**6.340.3.130 void CEGUI::Window::setArea (const URect & *area*)**

Set the window area.

Sets the area occupied by this window. The defined area is offset from the top-left corner of this windows parent window or from the top-left corner of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*area* URect describing the new area rectangle of the window area.

**6.340.3.131 void CEGUI::Window::setPosition (const UVector2 & *pos*)**

Set the window's position.

Sets the position of the area occupied by this window. The position is offset from the top-left corner of this windows parent window or from the top-left corner of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*pos* UVector2 describing the new position (top-left corner) of the window area.

**6.340.3.132 void CEGUI::Window::setXPosition (const UDim & *x*)**

Set the window's X position.

Sets the x position (left edge) of the area occupied by this window. The position is offset from the left edge of this windows parent window or from the left edge of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*x* UDim describing the new x position of the window area.

**6.340.3.133 void CEGUI::Window::setYPosition (const UDim & y)**

Set the window's Y position.

Sets the y position (top edge) of the area occupied by this window. The position is offset from the top edge of this windows parent window or from the top edge of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

y [UDim](#) describing the new y position of the window area.

**6.340.3.134 void CEGUI::Window::setSize (const UVector2 & size)**

Set the window's size.

Sets the size of the area occupied by this window.

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

size [UVector2](#) describing the new size of the window area.

**6.340.3.135 void CEGUI::Window::setWidth (const UDim & width)**

Set the window's width.

Sets the width of the area occupied by this window.

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

width [UDim](#) describing the new width of the window area.

**6.340.3.136 void CEGUI::Window::setHeight (const UDim & height)**

Set the window's height.

Sets the height of the area occupied by this window.



**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*height* [UDim](#) describing the new height of the window area.

**6.340.3.137 void CEGUI::Window::setMaxSize (const UVector2 & size)**

Set the window's maximum size.

Sets the maximum size that this windows area may occupy (whether size changes occur by user interaction, general system operation, or by direct setting by client code).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*size* [UVector2](#) describing the new maximum size of the window area.

**6.340.3.138 void CEGUI::Window::setMinSize (const UVector2 & size)**

Set the window's minimum size.

Sets the minimum size that this windows area may occupy (whether size changes occur by user interaction, general system operation, or by direct setting by client code).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Parameters:**

*size* [UVector2](#) describing the new minimum size of the window area.

**6.340.3.139 const URect & CEGUI::Window::getArea () const**

Return the windows area.

Returns the area occupied by this window. The defined area is offset from the top-left corner of this windows parent window or from the top-left corner of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[URect](#) describing the rectangle of the window area.

**6.340.3.140   const UVector2 & CEGUI::Window::getPosition () const**

Get the window's position.

Gets the position of the area occupied by this window. The position is offset from the top-left corner of this windows parent window or from the top-left corner of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UVector2](#) describing the position (top-left corner) of the window area.

**6.340.3.141   const UDim & CEGUI::Window::getXPosition () const**

Get the window's X position.

Gets the x position (left edge) of the area occupied by this window. The position is offset from the left edge of this windows parent window or from the left edge of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UDim](#) describing the x position of the window area.

**6.340.3.142   const UDim & CEGUI::Window::getYPosition () const**

Get the window's Y position.

Gets the y position (top edge) of the area occupied by this window. The position is offset from the top edge of this windows parent window or from the top edge of the display if this window has no parent (i.e. it is the root window).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UDim](#) describing the y position of the window area.

**6.340.3.143 UVector2 CEGUI::Window::getSize (void) const**

Get the window's size.

Gets the size of the area occupied by this window.

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UVector2](#) describing the size of the window area.

**6.340.3.144 UDim CEGUI::Window::getWidth (void) const**

Get the window's width.

Gets the width of the area occupied by this window.

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UDim](#) describing the width of the window area.

**6.340.3.145 UDim CEGUI::Window::getHeight (void) const**

Get the window's height.

Gets the height of the area occupied by this window.

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UDim](#) describing the height of the window area.

**6.340.3.146 const UVector2 & CEGUI::Window::getMaxSize () const**

Get the window's maximum size.

Gets the maximum size that this windows area may occupy (whether size changes occur by user interaction, general system operation, or by direct setting by client code).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UVector2](#) describing the maximum size of the window area.

**6.340.3.147 const UVector2 & CEGUI::Window::getMinSize () const**

Get the window's minimum size.

Gets the minimum size that this windows area may occupy (whether size changes occur by user interaction, general system operation, or by direct setting by client code).

**Note:**

This method makes use of "Unified Dimensions". These contain both parent relative and absolute pixel components, which are used in determining the final value used.

**Returns:**

[UVector2](#) describing the minimum size of the window area.

**6.340.3.148 void CEGUI::Window::render (void)**

Causes the [Window](#) object to render itself and all of it's attached children.

**Returns:**

Nothing

**6.340.3.149 void CEGUI::Window::update (float *elapsed*)**

Cause window to update itself and any attached children. Client code does not need to call this method; to ensure full, and proper updates, call the injectTimePulse methodname method provided by the [System](#) class.

**Note:**

The update order is such that 'this' window is updated prior to any child windows, this is so that child windows that access the parent in their update code get the correct updated state.

**Parameters:**

*elapsed* float value indicating the number of seconds passed since the last update.

**Returns:**

Nothing.

**6.340.3.150 void CEGUI::Window::writeXMLToStream (XMLSerializer & *xml\_stream*) const**  
[virtual]

Writes an xml representation of this window object to *out\_stream*.

**Parameters:**

*xml\_stream* Stream where xml data should be output.

**Returns:**

Nothing.

**6.340.3.151 void CEGUI::Window::setMousePassThroughEnabled (bool *setting*) [inline]**

Sets whether this window should ignore mouse events and pass them through to any windows behind it. In effect making the window transparent to the mouse.

**Parameters:**

*setting* true if mouse pass through is enabled. false if mouse pass through is not enabled.

**6.340.3.152 void CEGUI::Window::setWindowRenderer (const String & *name*)**

Assign the [WindowRenderer](#) to specify the Look'N'Feel specification to be used.

**Parameters:**

*name* The factory name of the [WindowRenderer](#) to use.

**Note:**

Once a window renderer has been assigned it is locked - as in cannot be changed.

**6.340.3.153 WindowRenderer \* CEGUI::Window::getWindowRenderer (void) const**

Get the currently assigned [WindowRenderer](#). (Look'N'Feel specification).

**Returns:**

A pointer to the assigned window renderer object. 0 if no window renderer is assigned.

**6.340.3.154 String CEGUI::Window::getWindowRendererName (void) const**

Get the factory name of the currently assigned [WindowRenderer](#). (Look'N'Feel specification).

**Returns:**

The factory name of the currently assigned [WindowRenderer](#). If no [WindowRenderer](#) is assigned an empty string is returned.

**6.340.3.155 void CEGUI::Window::setFalagardType (const String & *type*, const String & *rendererType* = "")**

Changes the widget's falagard type, thus changing its look'n'feel and optionally its renderer in the process.

**Parameters:**

*type* New look'n'feel of the widget

*type* New renderer of the widget

**6.340.3.156 void CEGUI::Window::setDragDropTarget (bool *setting*)**

Specifies whether this [Window](#) object will receive events generated by the drag and drop support in the system.

**Parameters:**

- setting* • true to enable the [Window](#) as a drag and drop target.
- false to disable the [Window](#) as a drag and drop target.

**6.340.3.157 void CEGUI::Window::onSized (WindowEventArgs & *e*)** [protected, virtual]

Handler called when the window's size changes.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::Listbox](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::ScrollablePane](#), and [CEGUI::Tree](#).

**6.340.3.158 void CEGUI::Window::onMoved (WindowEventArgs & *e*)** [protected, virtual]

Handler called when the window's position changes.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event. For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::DragContainer](#).

**6.340.3.159 void CEGUI::Window::onTextChanged (WindowEventArgs & *e*)** [protected, virtual]

Handler called when the window's text is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::Combobox](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::MenuItem](#), [CEGUI::MultiLineEditbox](#), [CEGUI::Spinner](#), and [CEGUI::Tooltip](#).

**6.340.3.160** `void CEGUI::Window::onFontChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's font is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::Combobox](#), [CEGUI::MultiColumnList](#), [CEGUI::Spinner](#), [CEGUI::TabControl](#), and [CEGUI::Titlebar](#).

**6.340.3.161** `void CEGUI::Window::onAlphaChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's alpha blend value is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::DragContainer](#), and [CEGUI::PopupMenu](#).

**6.340.3.162** `void CEGUI::Window::onIDChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the window's client assigned ID is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

**6.340.3.163** `void CEGUI::Window::onShown (WindowEventArgs & e)` [protected, virtual]

Handler called when the window is shown (made visible).

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::PopupMenu](#).

**6.340.3.164** `void CEGUI::Window::onHidden (WindowEventArgs & e)` [protected, virtual]

Handler called when the window is hidden.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::PopupMenu](#).

**6.340.3.165** `void CEGUI::Window::onEnabled (WindowEventArgs & e)` [protected, virtual]

Handler called when the window is enabled.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

**6.340.3.166** `void CEGUI::Window::onDisabled (WindowEventArgs & e)` [protected, virtual]

Handler called when the window is disabled.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

**6.340.3.167** `void CEGUI::Window::onClippingChanged (WindowEventArgs & e)`  
[protected, virtual]

Handler called when the window's setting for being clipped by it's parent is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::DragContainer](#).

**6.340.3.168** `void CEGUI::Window::onParentDestroyChanged (WindowEventArgs & e)`  
[protected, virtual]

Handler called when the window's setting for being destroyed automatically by it's parent is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.



**6.340.3.169** `void CEGUI::Window::onInheritsAlphaChanged (WindowEventArgs & e)`  
`[protected, virtual]`

Handler called when the window's setting for inheriting alpha-blending is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
 For this event the trigger window is always 'this'.

**6.340.3.170** `void CEGUI::Window::onAlwaysOnTopChanged (WindowEventArgs & e)`  
`[protected, virtual]`

Handler called when the window's always-on-top setting is changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
 For this event the trigger window is always 'this'.

**6.340.3.171** `void CEGUI::Window::onCaptureGained (WindowEventArgs & e)` `[protected, virtual]`

Handler called when this window gains capture of mouse inputs.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
 For this event the trigger window is always 'this'.

**6.340.3.172** `void CEGUI::Window::onCaptureLost (WindowEventArgs & e)` `[protected, virtual]`

Handler called when this window loses capture of mouse inputs.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
 For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::ButtonBase](#), [CEGUI::ComboDropList](#), [CEGUI::DragContainer](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::ListHeaderSegment](#), [CEGUI::MenuItem](#), [CEGUI::MultiLineEditbox](#), [CEGUI::Thumb](#), and [CEGUI::Titlebar](#).

**6.340.3.173** `void CEGUI::Window::onRenderingStarted (WindowEventArgs & e)`  
`[protected, virtual]`

Handler called when rendering for this window has started.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
 For this event the trigger window is always 'this'.

**6.340.3.174** `void CEGUI::Window::onRenderingEnded (WindowEventArgs & e)`  
[protected, virtual]

Handler called when rendering for this window has ended.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

**6.340.3.175** `void CEGUI::Window::onZChanged (WindowEventArgs & e)` [protected, virtual]

Handler called when the z-order position of this window has changed.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

**6.340.3.176** `void CEGUI::Window::onDestructionStarted (WindowEventArgs & e)`  
[protected, virtual]

Handler called when this window's destruction sequence has begun.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that triggered the event.  
For this event the trigger window is always 'this'.

Reimplemented in [CEGUI::PopupMenu](#).

**6.340.3.177** `void CEGUI::Window::onActivated (ActivationEventArgs & e)` [protected, virtual]

Handler called when this window has become the active window.

**Parameters:**

- e* [ActivationEventArgs](#) class whose 'otherWindow' field is set to the window that previously was active, or NULL for none.

Reimplemented in [CEGUI::Combobox](#), [CEGUI::ComboDropList](#), [CEGUI::FrameWindow](#), and [CEGUI::Spinner](#).

**6.340.3.178** `void CEGUI::Window::onDeactivated (ActivationEventArgs & e)` [protected, virtual]

Handler called when this window has lost input focus and has been deactivated.

**Parameters:**

- e* [ActivationEventArgs](#) object whose 'otherWindow' field is set to the window that has now become active, or NULL for none.

Reimplemented in [CEGUI::FrameWindow](#).

**6.340.3.179** `void CEGUI::Window::onParentSized (WindowEventArgs & e)` [protected, virtual]

Handler called when this window's parent window has been resized. If this window is the root / GUI Sheet window, this call will be made when the display size changes.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that caused the event; this is typically either this window's parent window, or NULL to indicate the screen size has changed.

Reimplemented in [CEGUI::ScrolledContainer](#).

**6.340.3.180** `void CEGUI::Window::onChildAdded (WindowEventArgs & e)` [protected, virtual]

Handler called when a child window is added to this window.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that has been added.

Reimplemented in [CEGUI::ScrolledContainer](#).

**6.340.3.181** `void CEGUI::Window::onChildRemoved (WindowEventArgs & e)` [protected, virtual]

Handler called when a child window is removed from this window.

**Parameters:**

- e* [WindowEventArgs](#) object whose 'window' pointer field is set to the window that has been removed.

Reimplemented in [CEGUI::ScrolledContainer](#).

**6.340.3.182** `void CEGUI::Window::onMouseEnters (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has entered this window's area.

**Parameters:**

- e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::Tooltip](#).

**6.340.3.183** `void CEGUI::Window::onMouseLeaves (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has left this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::ButtonBase](#), [CEGUI::ListHeaderSegment](#), and [CEGUI::MenuItem](#).

**6.340.3.184** `void CEGUI::Window::onMouseMove (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse cursor has been moved within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::ButtonBase](#), [CEGUI::ComboDropList](#), [CEGUI::DragContainer](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::Listbox](#), [CEGUI::ListHeaderSegment](#), [CEGUI::MenuItem](#), [CEGUI::MultiLineEditbox](#), [CEGUI::TabButton](#), [CEGUI::Thumb](#), [CEGUI::Titlebar](#), and [CEGUI::Tree](#).

**6.340.3.185** `void CEGUI::Window::onMouseWheel (MouseEventArgs & e)` [protected, virtual]

Handler called when the mouse wheel (z-axis) position changes within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::Listbox](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::ScrollablePane](#), [CEGUI::Scrollbar](#), [CEGUI::ScrolledItemListBase](#), [CEGUI::Slider](#), [CEGUI::TabButton](#), and [CEGUI::Tree](#).

**6.340.3.186** `void CEGUI::Window::onMouseButtonDown (MouseEventArgs & e)`  
[protected, virtual]

Handler called when a mouse button has been depressed within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::ButtonBase](#), [CEGUI::ComboDropList](#), [CEGUI::DragContainer](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::Listbox](#), [CEGUI::ListHeaderSegment](#), [CEGUI::MenuItem](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::PopupMenu](#), [CEGUI::Scrollbar](#), [CEGUI::Slider](#), [CEGUI::TabButton](#), [CEGUI::Thumb](#), [CEGUI::Titlebar](#), and [CEGUI::Tree](#).

**6.340.3.187** `void CEGUI::Window::onMouseButtonUp (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been released within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::ButtonBase](#), [CEGUI::Checkbox](#), [CEGUI::ComboDropList](#), [CEGUI::DragContainer](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::ListHeaderSegment](#), [CEGUI::MenuItem](#), [CEGUI::MultiLineEditbox](#), [CEGUI::PopupMenu](#), [CEGUI::PushButton](#), [CEGUI::RadioButton](#), [CEGUI::TabButton](#), and [CEGUI::Titlebar](#).

**6.340.3.188** `void CEGUI::Window::onMouseClicked (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been clicked (that is depressed and then released, within a specified time) within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::ItemEntry](#).

**6.340.3.189** `void CEGUI::Window::onMouseDoubleClicked (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been double-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::Editbox](#), [CEGUI::ListHeaderSegment](#), [CEGUI::MultiLineEditbox](#), and [CEGUI::Titlebar](#).

**6.340.3.190** `void CEGUI::Window::onMouseTripleClicked (MouseEventArgs & e)` [protected, virtual]

Handler called when a mouse button has been triple-clicked within this window's area.

**Parameters:**

*e* [MouseEventArgs](#) object. All fields are valid.

Reimplemented in [CEGUI::Editbox](#), and [CEGUI::MultiLineEditbox](#).

**6.340.3.191** `void CEGUI::Window::onKeyDown (KeyEventArgs & e)` [protected, virtual]

Handler called when a key as been depressed while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'scancode' field is set to the [Key::Scan](#) value representing the key that was pressed, and whose 'sysKeys' field represents the combination of [SystemKey](#) that were active when the event was generated.

Reimplemented in [CEGUI::Editbox](#), [CEGUI::ItemListbox](#), and [CEGUI::MultiLineEditbox](#).

**6.340.3.192** `void CEGUI::Window::onKeyUp (KeyEventArgs & e)` [protected, virtual]

Handler called when a key as been released while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'scancode' field is set to the [Key::Scan](#) value representing the key that was released, and whose 'sysKeys' field represents the combination of [SystemKey](#) that were active when the event was generated. All other fields should be considered as 'junk'.

**6.340.3.193** `void CEGUI::Window::onCharacter (KeyEventArgs & e)` [protected, virtual]

Handler called when a character-key has been pressed while this window has input focus.

**Parameters:**

- e* [KeyEventArgs](#) object whose 'codepoint' field is set to the Unicode code point (encoded as utf32) for the character typed, and whose 'sysKeys' field represents the combination of [SystemKey](#) that were active when the event was generated. All other fields should be considered as 'junk'.

Reimplemented in [CEGUI::Editbox](#), and [CEGUI::MultiLineEditbox](#).

**6.340.3.194** `void CEGUI::Window::onDragDropItemEnters (DragDropEventArgs & e)`  
[protected, virtual]

Handler called when a [DragContainer](#) is dragged over this window.

**Parameters:**

- e* [DragDropEventArgs](#) object initialised as follows:
  - window field is normally set to point to 'this' window.
  - dragDropItem is a pointer to a [DragContainer](#) window that triggered the event.

**6.340.3.195** `void CEGUI::Window::onDragDropItemLeaves (DragDropEventArgs & e)`  
[protected, virtual]

Handler called when a [DragContainer](#) is dragged over this window.

**Parameters:**

*e* [DragDropEventArgs](#) object initialised as follows:

- window field is normally set to point to 'this' window.
- dragDropItem is a pointer to a [DragContainer](#) window that triggered the event.

**6.340.3.196** `void CEGUI::Window::onDragDropItemDropped (DragDropEventArgs & e)`  
[protected, virtual]

Handler called when a [DragContainer](#) is dragged over this window.

**Parameters:**

*e* [DragDropEventArgs](#) object initialised as follows:

- window field is normally set to point to 'this' window.
- dragDropItem is a pointer to a [DragContainer](#) window that triggered the event.

**6.340.3.197** `void CEGUI::Window::onVerticalAlignmentChanged (WindowEventArgs & e)`  
[protected, virtual]

Handler called when the vertical alignment setting for the window is changed.

**Parameters:**

*e* [WindowEventArgs](#) object initialised as follows:

- window field is set to point to the [Window](#) object whos alignment has changed (typically 'this').

**6.340.3.198** `void CEGUI::Window::onHorizontalAlignmentChanged (WindowEventArgs & e)`  
[protected, virtual]

Handler called when the horizontal alignment setting for the window is changed.

**Parameters:**

*e* [WindowEventArgs](#) object initialised as follows:

- window field is set to point to the [Window](#) object whos alignment has changed (typically 'this').

**6.340.3.199** `void CEGUI::Window::onWindowRendererAttached (WindowEventArgs & e)`  
[protected, virtual]

Handler called when a new window renderer object is attached.

**Parameters:**

*e* [WindowEventArgs](#) object initialised as follows:

- window field is set to point to the [Window](#) object that just got a new window renderer attached. (typically 'this').

**6.340.3.200** `void CEGUI::Window::onWindowRendererDetached (WindowEventArgs & e)`  
[protected, virtual]

Handler called when the currently attached window renderer object is detached.

**Parameters:**

*e* [WindowEventArgs](#) object initialised as follows:

- window field is set to point to the [Window](#) object that just got lost its window renderer. (typically 'this').

**6.340.3.201** `void CEGUI::Window::updateSelf (float elapsed)` [protected, virtual]

Perform actual update processing for this [Window](#).

**Parameters:**

*elapsed* float value indicating the number of seconds elapsed since the last update call.

**Returns:**

Nothing.

Reimplemented in [CEGUI::PopupMenu](#), and [CEGUI::Tooltip](#).

**6.340.3.202** `void CEGUI::Window::drawSelf (float z)` [protected, virtual]

Perform the actual rendering for this [Window](#).

**Parameters:**

*z* float value specifying the base Z co-ordinate that should be used when rendering

**Returns:**

Nothing

Reimplemented in [CEGUI::ClippedContainer](#), [CEGUI::ScrolledContainer](#), and [CEGUI::TabControl](#).



**6.340.3.203** `virtual void CEGUI::Window::populateRenderCache ()` [`inline`, `protected`, `virtual`]

Update the rendering cache.

Populates the Window's [RenderCache](#) with imagery to be sent to the renderer.

Reimplemented in [CEGUI::Tree](#).

**6.340.3.204** `virtual bool CEGUI::Window::testClassName_impl (const String & class_name) const` [`inline`, `protected`, `virtual`]

Return whether this window was inherited from the given class name at some point in the inheritance hierarchy.

#### Parameters:

*class\_name* The class name that is to be checked.

#### Returns:

true if this window was inherited from *class\_name*. false if not.

Reimplemented in [CEGUI::ButtonBase](#), [CEGUI::Checkbox](#), [CEGUI::ClippedContainer](#), [CEGUI::Combobox](#), [CEGUI::ComboDropList](#), [CEGUI::DragContainer](#), [CEGUI::Editbox](#), [CEGUI::FrameWindow](#), [CEGUI::GroupBox](#), [CEGUI::GUISheet](#), [CEGUI::ItemEntry](#), [CEGUI::ItemListBase](#), [CEGUI::ItemListbox](#), [CEGUI::Listbox](#), [CEGUI::ListHeader](#), [CEGUI::ListHeaderSegment](#), [CEGUI::Menubar](#), [CEGUI::MenuBase](#), [CEGUI::MenuItem](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::PopupMenu](#), [CEGUI::ProgressBar](#), [CEGUI::PushButton](#), [CEGUI::RadioButton](#), [CEGUI::ScrollablePane](#), [CEGUI::Scrollbar](#), [CEGUI::ScrolledContainer](#), [CEGUI::ScrolledItemListBase](#), [CEGUI::Slider](#), [CEGUI::Spinner](#), [CEGUI::TabButton](#), [CEGUI::TabControl](#), [CEGUI::Thumb](#), [CEGUI::Titlebar](#), [CEGUI::Tooltip](#), and [CEGUI::Tree](#).

**6.340.3.205** `void CEGUI::Window::setParent (Window * parent)` [`protected`]

Set the parent window for this window object.

#### Parameters:

*parent* Pointer to a [Window](#) object that is to be assigned as the parent to this [Window](#).

#### Returns:

Nothing

**6.340.3.206** `bool CEGUI::Window::validateWindowRenderer (const String & name) const` [`protected`, `virtual`]

Function used in checking if a [WindowRenderer](#) is valid for this window.

#### Returns:

Returns true if the given [WindowRenderer](#) class name is valid for this window. False if not.

Reimplemented in [CEGUI::Editbox](#), [CEGUI::ItemEntry](#), [CEGUI::ItemListBase](#), [CEGUI::Listbox](#), [CEGUI::ListHeader](#), [CEGUI::MultiColumnList](#), [CEGUI::MultiLineEditbox](#), [CEGUI::ScrollablePane](#), [CEGUI::Scrollbar](#), [CEGUI::Slider](#), [CEGUI::TabControl](#), and [CEGUI::Tooltip](#).

**6.340.3.207** `bool CEGUI::Window::moveToFront_impl (bool wasClicked)` [protected, virtual]

Implements move to front behavior.

**Returns:**

Should return true if some action was taken, or false if there was nothing to be done.

**6.340.3.208** `bool CEGUI::Window::doRiseOnClick (void)` [protected]

Implementation of rise on click functionality.

**Returns:**

true if we did something, false if there was nothing to do.

**6.340.3.209** `void CEGUI::Window::setArea_impl (const UVector2 & pos, const UVector2 & size, bool topLeftSizing = false, bool fireEvents = true)` [protected]

Implementation method to modify window area while correctly applying min / max size processing, and firing any appropriate events.

/note This is the implementation function for setting size and position. In order to simplify area management, from this point on, all modifications to window size and position (area rect) should come through here.

/param pos [UVector2](#) object describing the new area position.

/param size [UVector2](#) object describing the new area size.

/param topLeftSizing

- true to indicate the the operation is a sizing operation on the top and/or left edges of the area, and so window movement should be inhibited if size is at max or min.
- false to indicate the operation is not a strict sizing operation on the top and/or left edges and that the window position may change as required

/param fireEvents

- true if events should be fired as normal.
- false to inhibit firing of events (required, for example, if you need to call this from the onSize/onMove handlers).

**6.340.3.210** `void CEGUI::Window::addWindowToDrawList (Window & wnd, bool at_back = false) [protected]`

Add the given window to the drawing list at an appropriate position for it's settings and the required direction. Basically, when *at\_back* is false, the window will appear in front of all other windows with the same 'always on top' setting. When *at\_back* is true, the window will appear behind all other windows with the same 'always on top' setting.

**Parameters:**

*wnd* [Window](#) object to be added to the drawing list.

*at\_back* Indicates whether the window should be placed at the back of other windows in the same group. If this is false, the window is placed in front of other windows in the group.

**Returns:**

Nothing.

**6.340.3.211** `void CEGUI::Window::removeWindowFromDrawList (const Window & wnd) [protected]`

Removes the window from the drawing list. If the window is not attached to the drawing list then nothing happens.

**Parameters:**

*wnd* [Window](#) object to be removed from the drawing list.

**Returns:**

Nothing.

**6.340.3.212** `bool CEGUI::Window::isTopOfZOrder () const [protected]`

Return whether the window is at the top of the Z-Order. This will correctly take into account 'Always on top' windows as needed.

**Returns:**

- true if the [Window](#) is at the top of the z-order in relation to sibling windows with the same 'always on top' setting.
- false if the [Window](#) is not at the top of the z-order in relation to sibling windows with the same 'always on top' setting.

## 6.340.4 Member Data Documentation

**6.340.4.1** `const String CEGUI::Window::EventWindowUpdated [static]`

[Event](#) to signal the window is being updated. Used by lua system!

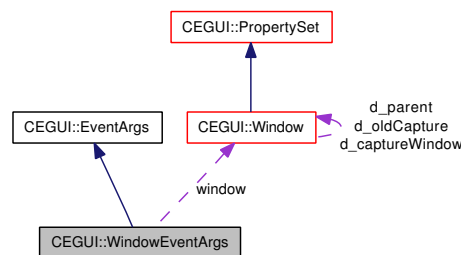
## 6.341 CEGUI::WindowEventArgs Class Reference

[EventArgs](#) based class that is used for objects passed to handlers triggered for events concerning some [Window](#) object.

Inherits [CEGUI::EventArgs](#).

Inherited by [CEGUI::ActivationEventArgs](#), [CEGUI::DragDropEventArgs](#), [CEGUI::HeaderSequenceEventArgs](#), [CEGUI::KeyEventArgs](#), [CEGUI::MouseEventArgs](#), [CEGUI::TreeEventArgs](#), and [CEGUI::UpdateEventArgs](#).

Collaboration diagram for CEGUI::WindowEventArgs:



### Public Member Functions

- **WindowEventArgs** ([Window](#) \*wnd)

### Public Attributes

- [Window](#) \* **window**  
*pointer to a [Window](#) object of relevance to the event.*

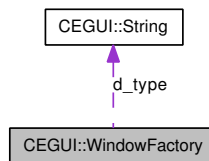
#### 6.341.1 Detailed Description

[EventArgs](#) based class that is used for objects passed to handlers triggered for events concerning some [Window](#) object.

## 6.342 CEGUI::WindowFactory Class Reference

Abstract class that defines the required interface for all [WindowFactory](#) objects.

Collaboration diagram for CEGUI::WindowFactory:



### Public Member Functions

- virtual [Window](#) \* [createWindow](#) (const [String](#) &name)=0  
*Create a new [Window](#) object of whatever type this [WindowFactory](#) produces.*
- virtual void [destroyWindow](#) ([Window](#) \*window)=0  
*Destroys the given [Window](#) object.*
- const [String](#) & [getTypeName](#) (void) const  
*Get the string that describes the type of [Window](#) object this [WindowFactory](#) produces.*

### Protected Member Functions

- [WindowFactory](#) (const [String](#) &type)

### Protected Attributes

- [String](#) [d\\_type](#)  
*[String](#) holding the type of object created by this factory.*

#### 6.342.1 Detailed Description

Abstract class that defines the required interface for all [WindowFactory](#) objects.

A [WindowFactory](#) is used to create and destroy windows of a specific type. For every type of [Window](#) object within the system (widgets, dialogs, movable windows etc) there must be an associated [WindowFactory](#) registered with the [WindowManager](#) so that the system knows how to create and destroy those types of [Window](#) base object.

#### 6.342.2 Member Function Documentation

- 6.342.2.1** virtual [Window](#)\* [CEGUI::WindowFactory::createWindow](#) (const [String](#) & *name*)  
[pure virtual]

Create a new [Window](#) object of whatever type this [WindowFactory](#) produces.

**Parameters:**

*name* A unique name that is to be assigned to the newly created [Window](#) object

**Returns:**

Pointer to the new [Window](#) object.

**6.342.2.2** `virtual void CEGUI::WindowFactory::destroyWindow (Window * window)` `[pure virtual]`

Destroys the given [Window](#) object.

**Parameters:**

*window* Pointer to the [Window](#) object to be destroyed.

**Returns:**

Nothing.

**6.342.2.3** `const String& CEGUI::WindowFactory::getTypeName (void) const` `[inline]`

Get the string that describes the type of [Window](#) object this [WindowFactory](#) produces.

**Returns:**

[String](#) object that contains the unique [Window](#) object type produced by this [WindowFactory](#)

## 6.343 CEGUI::WindowFactoryManager Class Reference

Class that manages [WindowFactory](#) objects.

### Public Types

- typedef [ConstBaseIterator](#)< WindowFactoryRegistry > **WindowFactoryIterator**
- typedef [ConstBaseIterator](#)< TypeAliasRegistry > **TypeAliasIterator**
- typedef [ConstBaseIterator](#)< FalagardMapRegistry > **FalagardMappingIterator**

### Public Member Functions

- [WindowFactoryManager](#) (void)  
*Constructs a new [WindowFactoryManager](#) object.*
- [~WindowFactoryManager](#) (void)  
*Destructor for [WindowFactoryManager](#) objects.*
- void [addFactory](#) ([WindowFactory](#) \*factory)  
*Adds a new [WindowFactory](#) to the list of registered factories.*
- void [removeFactory](#) (const [String](#) &name)  
*Removes a [WindowFactory](#) from the list of registered factories.*
- void [removeFactory](#) ([WindowFactory](#) \*factory)  
*Removes a [WindowFactory](#) from the list of registered factories.*
- void [removeAllFactories](#) (void)  
*Remove all [WindowFactory](#) objects from the list.*
- [WindowFactory](#) \* [getFactory](#) (const [String](#) &type) const  
*Return a pointer to the specified [WindowFactory](#) object.*
- bool [isFactoryPresent](#) (const [String](#) &name) const  
*Checks the list of registered [WindowFactory](#) objects, aliases, and falagard mapped types for one which can create [Window](#) objects of the specified type.*
- void [addWindowTypeAlias](#) (const [String](#) &aliasName, const [String](#) &targetType)  
*Adds an alias for a current window type.*
- void [removeWindowTypeAlias](#) (const [String](#) &aliasName, const [String](#) &targetType)  
*Remove the specified alias mapping. If the alias mapping does not exist, nothing happens.*
- void [addFalagardWindowMapping](#) (const [String](#) &newType, const [String](#) &targetType, const [String](#) &lookName, const [String](#) &renderer)  
*Add a mapping for a falagard based window.*
- void [removeFalagardWindowMapping](#) (const [String](#) &type)  
*Remove the specified falagard type mapping if it exists.*

- `bool isFalagardMappedType (const String &type) const`  
*Return whether the given type is a falagard mapped type.*
- `const String & getMappedLookForType (const String &type) const`  
*Return the name of the LookN'Feel assigned to the specified window mapping.*
- `const String & getMappedRendererForType (const String &type) const`  
*Return the name of the WindowRenderer assigned to the specified window mapping.*
- `String getDereferencedAliasType (const String &type) const`  
*Use the alias system, where required, to 'de-reference' the specified type to an actual window type that can be created directly (that being either a concrete window type, or a falagard mapped type).*
- `const FalagardWindowMapping & getFalagardMappingForType (const String &type) const`  
*Return the FalagardWindowMapping for the specified window mapping type.*
- `WindowFactoryIterator getIterator (void) const`  
*Return a WindowFactoryManager::WindowFactoryIterator object to iterate over the available WindowFactory types.*
- `TypeAliasIterator getAliasIterator (void) const`  
*Return a WindowFactoryManager::TypeAliasIterator object to iterate over the defined aliases for window types.*
- `FalagardMappingIterator getFalagardMappingIterator () const`  
*Return a WindowFactoryManager::FalagardMappingIterator object to iterate over the defined falagard window mappings.*

## Classes

- class `AliasTargetStack`  
*Class used to track active alias targets for Window factory types.*
- struct `FalagardWindowMapping`  
*struct used to hold mapping information required to create a falagard based window.*

### 6.343.1 Detailed Description

Class that manages `WindowFactory` objects.

#### Todo

I think we could clean up the mapping stuff a bit. Possibly make it more generic now with the window renderers and all.



## 6.343.2 Member Function Documentation

### 6.343.2.1 void CEGUI::WindowFactoryManager::addFactory (WindowFactory \* *factory*)

Adds a new [WindowFactory](#) to the list of registered factories.

**Parameters:**

*factory* Pointer to the [WindowFactory](#) to be added to the [WindowManager](#).

**Returns:**

Nothing

**Exceptions:**

[NullObjectException](#) *factory* was null.

[AlreadyExistsException](#) *factory* provided a [Window](#) type name which is in use by another registered [WindowFactory](#).

### 6.343.2.2 void CEGUI::WindowFactoryManager::removeFactory (const String & *name*)

Removes a [WindowFactory](#) from the list of registered factories.

**Note:**

The [WindowFactory](#) object is not destroyed (since it was created externally), instead it is just removed from the list.

**Parameters:**

*name* [String](#) which holds the name (technically, [Window](#) type name) of the [WindowFactory](#) to be removed. If *name* is not in the list, no error occurs (nothing happens).

**Returns:**

Nothing

### 6.343.2.3 void CEGUI::WindowFactoryManager::removeFactory (WindowFactory \* *factory*)

Removes a [WindowFactory](#) from the list of registered factories.

**Note:**

The [WindowFactory](#) object is not destroyed (since it was created externally), instead it is just removed from the list.

**Parameters:**

*factory* Pointer to the factory object to be removed. If *factory* is null, or if no such [WindowFactory](#) is in the list, no error occurs (nothing happens).

**Returns:**

Nothing

**6.343.2.4 void CEGUI::WindowFactoryManager::removeAllFactories (void) [inline]**

Remove all [WindowFactory](#) objects from the list.

**Returns:**

Nothing

**6.343.2.5 WindowFactory \* CEGUI::WindowFactoryManager::getFactory (const String & type) const**

Return a pointer to the specified [WindowFactory](#) object.

**Parameters:**

*type* [String](#) holding the [Window](#) object type to return the [WindowFactory](#) for.

**Returns:**

Pointer to the [WindowFactory](#) object that creates Windows of the type *type*.

**Exceptions:**

[UnknownObjectException](#) No [WindowFactory](#) object for [Window](#) objects of type *type* was found.

**6.343.2.6 bool CEGUI::WindowFactoryManager::isFactoryPresent (const String & name) const**

Checks the list of registered [WindowFactory](#) objects, aliases, and falagard mapped types for one which can create [Window](#) objects of the specified type.

**Parameters:**

*name* [String](#) containing the [Window](#) type name to check for.

**Returns:**

- true if a [WindowFactory](#), alias, or falagard mapping for [Window](#) objects of type *name* is registered.
- false if the system knows nothing about windows of type *name*.

**6.343.2.7 void CEGUI::WindowFactoryManager::addWindowTypeAlias (const String & aliasName, const String & targetType)**

Adds an alias for a current window type.

This method allows you to create an alias for a specified window type. This means that you can then use either name as the type parameter when creating a window.

**Note:**

You need to be careful using this system. Creating an alias using a name that already exists will replace the previous mapping for that alias. Each alias name maintains a stack, which means that it is possible to remove an alias and have the previous alias restored. The windows created via an alias use the real type, so removing an alias after window creation is always safe (i.e. it is not the same as removing a real factory, which would cause an exception when trying to destroy a window with a missing factory).

**Parameters:**

*aliasName* [String](#) object holding the alias name. That is the name that *targetType* will also be known as from now on.

*targetType* [String](#) object holding the type window type name that is to be aliased. This type must already exist.

**Returns:**

Nothing.

**Exceptions:**

[UnknownObjectException](#) thrown if *targetType* is not known within the system.

**6.343.2.8 void CEGUI::WindowFactoryManager::removeWindowTypeAlias (const String & aliasName, const String & targetType)**

Remove the specified alias mapping. If the alias mapping does not exist, nothing happens.

**Note:**

You are required to supply both the alias and target names because there may exist more than one entry for a given alias - therefore you are required to be explicit about which alias is to be removed.

**Parameters:**

*aliasName* [String](#) object holding the alias name.

*targetType* [String](#) object holding the type window type name that was aliased.

**Returns:**

Nothing.

**6.343.2.9 void CEGUI::WindowFactoryManager::addFalagardWindowMapping (const String & newType, const String & targetType, const String & lookName, const String & renderer)**

Add a mapping for a falagard based window.

This function creates maps a target window type and target 'look' name onto a registered window type, thus allowing the usual window creation interface to be used to create windows that require extra information to full initialise themselves.

**Note:**

These mappings support 'late binding' to the target window type, as such the type indicated by *targetType* need not exist in the system until attempting to create a [Window](#) using the type.

Also note that creating a mapping for an existing type will replace any previous mapping for that same type.

**Parameters:**

*newType* The type name that will be used to create windows using the target type and look.

*targetType* The base window type.

*lookName* The name of the 'look' that will be used by windows of this type.

*renderer* The type of window renderer to assign for windows of this type.

**Returns:**

Nothing.

**6.343.2.10 void CEGUI::WindowFactoryManager::removeFalagardWindowMapping (const String & type)**

Remove the specified falagard type mapping if it exists.

**Returns:**

Nothing.

**6.343.2.11 bool CEGUI::WindowFactoryManager::isFalagardMappedType (const String & type) const**

Return whether the given type is a falagard mapped type.

**Parameters:**

*type* Name of a window type.

**Returns:**

- true if the requested type is a Falagard mapped window type.
- false if the requested type is a normal [WindowFactory](#) (or alias), or if the type does not exist.

**6.343.2.12 const String & CEGUI::WindowFactoryManager::getMappedLookForType (const String & type) const**

Return the name of the LookN'Feel assigned to the specified window mapping.

**Parameters:**

*type* Name of a window type. The window type referenced should be a falagard mapped type.

**Returns:**

[String](#) object holding the name of the look mapped for the requested type.

**Exceptions:**

[InvalidRequestException](#) thrown if *type* is not a falagard mapping type (or maybe the type didn't exist).

**6.343.2.13** `const String & CEGUI::WindowFactoryManager::getMappedRendererForType (const String & type) const`

Return the name of the [WindowRenderer](#) assigned to the specified window mapping.

**Parameters:**

*type* Name of a window type. The window type referenced should be a falagard mapped type.

**Returns:**

[String](#) object holding the name of the window renderer mapped for the requested type.

**Exceptions:**

[InvalidRequestException](#) thrown if *type* is not a falagard mapping type (or maybe the type didn't exist).

**6.343.2.14** `String CEGUI::WindowFactoryManager::getDereferencedAliasType (const String & type) const`

Use the alias system, where required, to 'de-reference' the specified type to an actual window type that can be created directly (that being either a concrete window type, or a falagard mapped type).

**Note:**

Even though implied by the above description, this method does not check that a factory for the final type exists; we simply say that the returned type is not an alias for some other type.

**Parameters:**

*type* [String](#) describing the type to be de-referenced.

**Returns:**

[String](#) object holding a type for a window that can be created directly; that is, a type that does not describe an alias to some other type.

**6.343.2.15** `const WindowFactoryManager::FalagardWindowMapping & CEGUI::WindowFactoryManager::getFalagardMappingForType (const String & type) const`

Return the [FalagardWindowMapping](#) for the specified window mapping *type*.

**Parameters:**

*type* Name of a window type. The window type referenced should be a falagard mapped type.

**Returns:**

[FalagardWindowMapping](#) object describing the falagard mapping.

**Exceptions:**

[InvalidRequestException](#) thrown if *type* is not a falagard mapping type (or maybe the type didn't exist).

## 6.344 CEGUI::WindowFactoryManager::AliasTargetStack Class Reference

Class used to track active alias targets for [Window](#) factory types.

### Public Member Functions

- [AliasTargetStack](#) (void)  
*Constructor for WindowAliasTargetStack objects.*
- [~AliasTargetStack](#) (void)  
*Destructor for WindowAliasTargetStack objects.*
- const [String](#) & [getActiveTarget](#) (void) const  
*Return a [String](#) holding the current target type for this stack.*
- uint [getStackedTargetCount](#) (void) const  
*Return the number of stacked target types in the stack.*

### Friends

- class [WindowFactoryManager](#)

### 6.344.1 Detailed Description

Class used to track active alias targets for [Window](#) factory types.

### 6.344.2 Member Function Documentation

#### 6.344.2.1 const [String](#) & CEGUI::WindowFactoryManager::AliasTargetStack::getActiveTarget (void) const

Return a [String](#) holding the current target type for this stack.

#### Returns:

reference to a [String](#) object holding the currently active target type name for this stack.

#### 6.344.2.2 uint CEGUI::WindowFactoryManager::AliasTargetStack::getStackedTargetCount (void) const

Return the number of stacked target types in the stack.

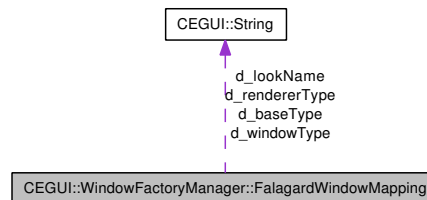
#### Returns:

number of target types stacked for this alias.

## 6.345 CEGUI::WindowFactoryManager::FalagardWindowMapping Struct Reference

struct used to hold mapping information required to create a falagard based window.

Collaboration diagram for CEGUI::WindowFactoryManager::FalagardWindowMapping:



### Public Attributes

- [String](#) d\_windowType
- [String](#) d\_lookName
- [String](#) d\_baseType
- [String](#) d\_rendererType

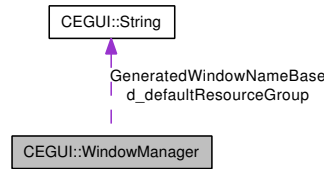
### 6.345.1 Detailed Description

struct used to hold mapping information required to create a falagard based window.

## 6.346 CEGUI::WindowManager Class Reference

The [WindowManager](#) class describes an object that manages creation and lifetime of [Window](#) objects.

Collaboration diagram for CEGUI::WindowManager:



### Public Types

- typedef [ConstBaseIterator](#)< WindowRegistry > **WindowIterator**

### Public Member Functions

- typedef bool [PropertyCallback](#) ([Window](#) \*window, [String](#) &propname, [String](#) &propvalue, void \*userdata)  
*Function type that is used as a callback when loading layouts from XML; the function is called for each [Property](#) element encountered.*
- [WindowManager](#) (void)  
*Constructs a new [WindowManager](#) object.*
- [~WindowManager](#) (void)  
*Destructor for [WindowManager](#) objects.*
- [Window](#) \* [createWindow](#) (const [String](#) &type, const [String](#) &name="", const [String](#) &prefix="")  
*Creates a new [Window](#) object of the specified type, and gives it the specified unique name.*
- void [destroyWindow](#) ([Window](#) \*window)  
*Destroy the specified [Window](#) object.*
- void [destroyWindow](#) (const [String](#) &window)  
*Destroy the specified [Window](#) object.*
- [Window](#) \* [getWindow](#) (const [String](#) &name) const  
*Return a pointer to the specified [Window](#) object.*
- bool [isWindowPresent](#) (const [String](#) &name) const  
*Examines the list of [Window](#) objects to see if one exists with the given name.*
- void [destroyAllWindows](#) (void)  
*Destroys all [Window](#) objects within the system.*
- [Window](#) \* [loadWindowLayout](#) (const [String](#) &filename, const [String](#) &name\_prefix="", const [String](#) &resourceGroup="", [PropertyCallback](#) \*callback=0, void \*userdata=0)



*Creates a set of windows (a Gui layout) from the information in the specified XML file.*

- **Window \* loadWindowLayout** (const **String** &filename, bool generateRandomPrefix)
- bool **isDeadPoolEmpty** (void) const

*Return whether the window dead pool is empty.*

- void **cleanDeadPool** (void)

*Permanently destroys any windows placed in the dead pool.*

- void **writeWindowLayoutToStream** (const **Window** &window, **OutStream** &out\_stream, bool writeParent=false) const

*Writes a full XML window layout, starting at the given Window to the given OutStream.*

- void **writeWindowLayoutToStream** (const **String** &window, **OutStream** &out\_stream, bool writeParent=false) const

*Writes a full XML window layout, starting at the given Window to the given OutStream.*

- void **renameWindow** (const **String** &window, const **String** &new\_name)

*Rename a window.*

- void **renameWindow** (**Window** \*window, const **String** &new\_name)

*Rename a window.*

- **WindowIterator getIterator** (void) const

*Return a WindowManager::WindowIterator object to iterate over the currently defined Windows.*

- void **DEBUG\_dumpWindowNames** (**String** zone)

*Outputs the names of ALL existing windows to log (DEBUG function).*

## Static Public Member Functions

- static const **String** & **getDefaultResourceGroup** ()

*Returns the default resource group currently set for layouts.*

- static void **setDefaultResourceGroup** (const **String** &resourceGroup)

*Sets the default resource group to be used when loading layouts.*

## Static Public Attributes

- static const **String** **GeneratedWindowNameBase**

*Base name to use for generated window names.*

### 6.346.1 Detailed Description

The [WindowManager](#) class describes an object that manages creation and lifetime of [Window](#) objects.

The [WindowManager](#) is the means by which [Window](#) objects are created and destroyed. For each subclass of [Window](#) that is to be created, there must exist a [WindowFactory](#) object which is registered with the [WindowFactoryManager](#). Additionally, the [WindowManager](#) tracks every [Window](#) object created, and can be used to access those [Window](#) objects by name.

### 6.346.2 Constructor & Destructor Documentation

#### 6.346.2.1 CEGUI::WindowManager::WindowManager (void)

Constructs a new [WindowManager](#) object.

NB: Client code should not create [WindowManager](#) objects - they are of limited use to you! The intended pattern of access is to get a pointer to the GUI system's [WindowManager](#) via the [System](#) object, and use that.

#### 6.346.2.2 CEGUI::WindowManager::~~WindowManager (void)

Destructor for [WindowManager](#) objects.

This will properly destroy all remaining [Window](#) objects. Note that [WindowFactory](#) objects will not be destroyed (since they are owned by whoever created them).

### 6.346.3 Member Function Documentation

#### 6.346.3.1 typedef bool CEGUI::WindowManager::PropertyCallback (Window \* *window*, String & *propname*, String & *propvalue*, void \* *userdata*)

Function type that is used as a callback when loading layouts from XML; the function is called for each [Property](#) element encountered.

##### Parameters:

- window* [Window](#) object that the property is to be applied to.
- propname* [String](#) holding the name of the property that is being set.
- propvalue* [String](#) holding the new value that will be applied to the property specified by /a *propname*.
- userdata* Some client code supplied data.

##### Returns:

- true if the property should be set.
- false if the property should not be set,

#### 6.346.3.2 Window \* CEGUI::WindowManager::createWindow (const String & *type*, const String & *name* = "", const String & *prefix* = "")

Creates a new [Window](#) object of the specified type, and gives it the specified unique name.

**Parameters:**

*type* [String](#) that describes the type of [Window](#) to be created. A valid [WindowFactory](#) for the specified type must be registered.

*name* [String](#) that holds a unique name that is to be given to the new window. If this string is empty (""), a name will be generated for the window.

**Returns:**

Pointer to the newly created [Window](#) object.

**Exceptions:**

[AlreadyExistsException](#) A [Window](#) object with the name *name* already exists.

[UnknownObjectException](#) No [WindowFactory](#) is registered for type [Window](#) objects.

[GenericException](#) Some other error occurred ([Exception](#) message has details).

**6.346.3.3 void CEGUI::WindowManager::destroyWindow (Window \* window)**

Destroy the specified [Window](#) object.

**Parameters:**

*window* Pointer to the [Window](#) object to be destroyed. If the *window* is null, or is not recognised, nothing happens.

**Returns:**

Nothing

**Exceptions:**

[InvalidRequestException](#) Can be thrown if the [WindowFactory](#) for *window*'s object type was removed.

**6.346.3.4 void CEGUI::WindowManager::destroyWindow (const String & window)**

Destroy the specified [Window](#) object.

**Parameters:**

*window* [String](#) containing the name of the [Window](#) object to be destroyed. If *window* is not recognised, nothing happens.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Can be thrown if the [WindowFactory](#) for *window*'s object type was removed.

**6.346.3.5 Window \* CEGUI::WindowManager::getWindow (const String & name) const**

Return a pointer to the specified [Window](#) object.

**Parameters:**

*name* [String](#) holding the name of the [Window](#) object to be returned.

**Returns:**

Pointer to the [Window](#) object with the name *name*.

**Exceptions:**

[UnknownObjectException](#) No [Window](#) object with a name matching *name* was found.

**6.346.3.6 bool CEGUI::WindowManager::isWindowPresent (const String & name) const**

Examines the list of [Window](#) objects to see if one exists with the given name.

**Parameters:**

*name* [String](#) holding the name of the [Window](#) object to look for.

**Returns:**

true if a [Window](#) object was found with a name matching *name*. false if no matching [Window](#) object was found.

**6.346.3.7 void CEGUI::WindowManager::destroyAllWindows (void)**

Destroys all [Window](#) objects within the system.

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown if the [WindowFactory](#) for any [Window](#) object type has been removed.

**6.346.3.8 Window \* CEGUI::WindowManager::loadWindowLayout (const String & filename, const String & name\_prefix = "", const String & resourceGroup = "", PropertyCallback \* callback = 0, void \* userdata = 0)**

Creates a set of windows (a Gui layout) from the information in the specified XML file.

**Parameters:**

*filename* [String](#) object holding the filename of the XML file to be processed.

**name\_prefix** [String](#) object holding the prefix that is to be used when creating the windows in the layout file, this function allows a layout to be loaded multiple times without having name clashes. Note that if you use this facility, then all windows defined within the layout must have names assigned; you currently can not use this feature in combination with automatically generated window names.

**resourceGroup** Resource group identifier to be passed to the resource provider when loading the layout file.

**callback** PropertyCallback function to be called for each [Property](#) element loaded from the layout. This is called prior to the property value being applied to the window enabling client code manipulation of properties.

**userdata** Client code data pointer passed to the PropertyCallback function.

**Returns:**

Pointer to the root [Window](#) object defined in the layout.

**Exceptions:**

[FileIOException](#) thrown if something goes wrong while processing the file *filename*.

[InvalidRequestException](#) thrown if *filename* appears to be invalid.

### 6.346.3.9 bool CEGUI::WindowManager::isDeadPoolEmpty (void) const

Return whether the window dead pool is empty.

**Returns:**

- true if there are no windows in the dead pool.
- false if the dead pool contains  $\geq 1$  window awaiting destruction.

### 6.346.3.10 void CEGUI::WindowManager::cleanDeadPool (void)

Permanently destroys any windows placed in the dead pool.

**Note:**

It is probably not a good idea to call this from a [Window](#) based event handler if the specific window has been or is being destroyed.

**Returns:**

Nothing.

### 6.346.3.11 void CEGUI::WindowManager::writeWindowLayoutToStream (const Window & window, OutStream & out\_stream, bool writeParent = false) const

Writes a full XML window layout, starting at the given [Window](#) to the given OutStream.

**Parameters:**

**window** [Window](#) object to become the root of the layout.

*out\_stream* OutStream (std::ostream based) object where data is to be sent.

*writeParent* If the starting window has a parent window, specifies whether to write the parent name into the Parent attribute of the GUILayout XML element.

**Returns:**

Nothing.

**6.346.3.12 void CEGUI::WindowManager::writeWindowLayoutToStream (const String & window, OutStream & out\_stream, bool writeParent = false) const**

Writes a full XML window layout, starting at the given [Window](#) to the given OutStream.

**Parameters:**

*window* [String](#) holding the name of the [Window](#) object to become the root of the layout.

*out\_stream* OutStream (std::ostream based) object where data is to be sent.

*writeParent* If the starting window has a parent window, specifies whether to write the parent name into the Parent attribute of the GUILayout XML element.

**Returns:**

Nothing.

**6.346.3.13 void CEGUI::WindowManager::renameWindow (const String & window, const String & new\_name)**

Rename a window.

**Parameters:**

*window* [String](#) holding the current name of the window to be renamed.

*new\_name* [String](#) holding the new name for the window

**Exceptions:**

[UnknownObjectException](#) thrown if *window* is not known in the system.

[AlreadyExistsException](#) thrown if a [Window](#) named *new\_name* already exists.

**6.346.3.14 void CEGUI::WindowManager::renameWindow (Window \* window, const String & new\_name)**

Rename a window.

**Parameters:**

*window* Pointer to the window to be renamed.

*new\_name* [String](#) holding the new name for the window

**Exceptions:**

[AlreadyExistsException](#) thrown if a [Window](#) named *new\_name* already exists.

**6.346.3.15 static const String& CEGUI::WindowManager::getDefaultResourceGroup (void)**  
[inline, static]

Returns the default resource group currently set for layouts.

**Returns:**

[String](#) describing the default resource group identifier that will be used when loading layouts.

**6.346.3.16 static void CEGUI::WindowManager::setDefaultResourceGroup (const String &resourceGroup)** [inline, static]

Sets the default resource group to be used when loading layouts.

**Parameters:**

*resourceGroup* [String](#) describing the default resource group identifier to be used.

**Returns:**

Nothing.

**6.346.3.17 void CEGUI::WindowManager::DEBUG\_dumpWindowNames (String zone)**

Outputs the names of ALL existing windows to log (DEBUG function).

**Parameters:**

*zone* Helper string that can specify where the name dump was made (e.g. "BEFORE FRAME DELETION").

**Returns:**

Nothing.

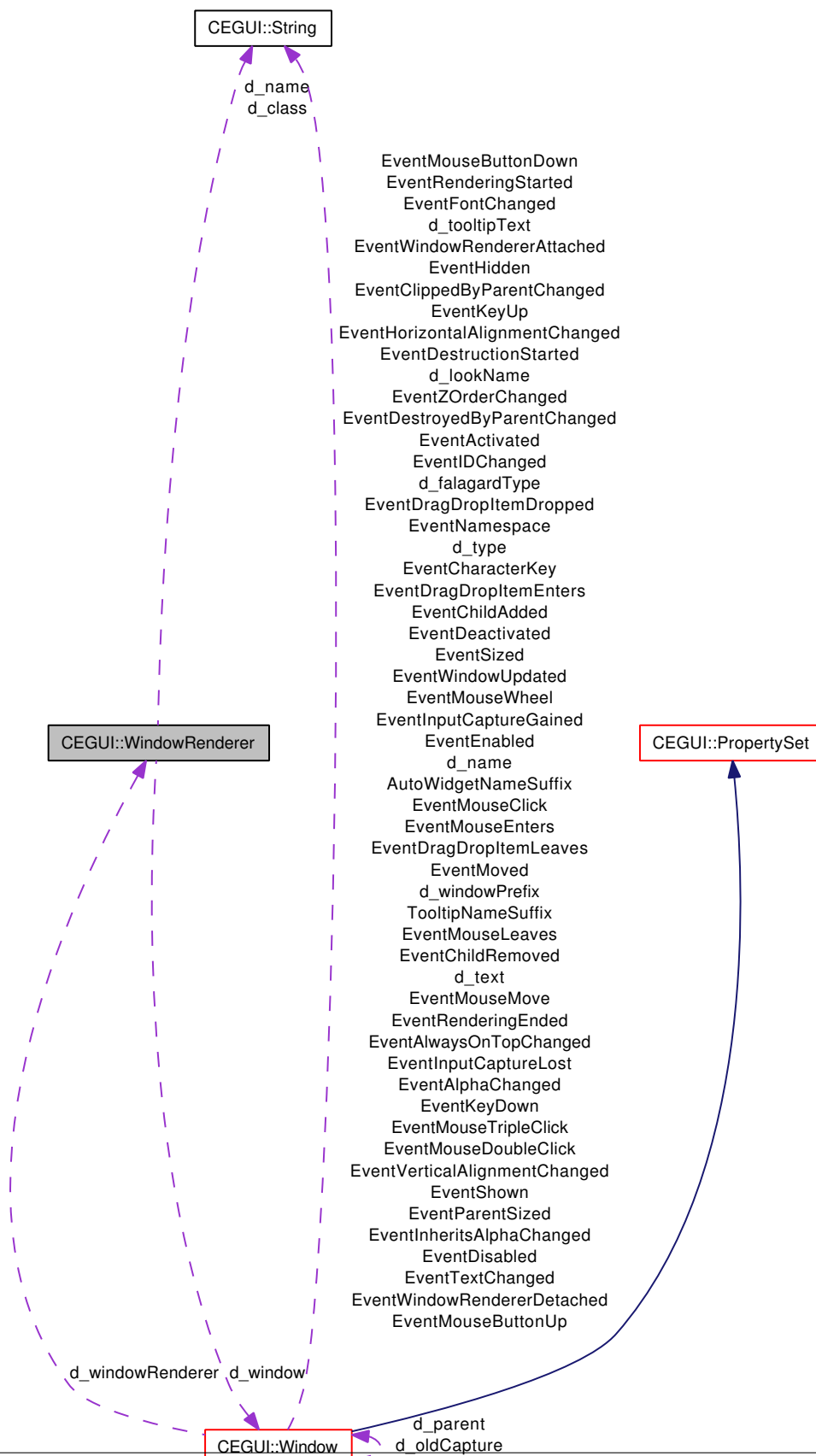
## 6.347 CEGUI::WindowRenderer Class Reference

Base-class for the assignable [WindowRenderer](#) object.

Inherited by [CEGUI::EditboxWindowRenderer](#), [CEGUI::ItemEntryWindowRenderer](#),  
[CEGUI::ItemListBaseWindowRenderer](#), [CEGUI::ListboxWindowRenderer](#),  
[CEGUI::ListHeaderWindowRenderer](#), [CEGUI::MultiColumnListWindowRenderer](#),  
[CEGUI::MultiLineEditboxWindowRenderer](#), [CEGUI::ScrollablePaneWindowRenderer](#),  
[CEGUI::ScrollbarWindowRenderer](#), [CEGUI::SliderWindowRenderer](#), [CEGUI::TabControlWindowRenderer](#),  
and [CEGUI::TooltipWindowRenderer](#).



Collaboration diagram for CEGUI::WindowRenderer:



## Public Member Functions

- **WindowRenderer** (const **String** &name, const **String** &class\_name="Window")  
*Constructor.*
- virtual **~WindowRenderer** ()  
*Destructor.*
- virtual void **render** ()=0  
*Populate render cache.*
- const **String** & **getName** () const  
*Returns the factory type name of this window renderer.*
- **Window** \* **getWindow** () const  
*Get the window this windowrenderer is attached to.*
- const **String** & **getClass** () const  
*Get the "minimum" **Window** class this renderer requires.*
- const **WidgetLookAndFeel** & **getLookAndFeel** () const  
*Get the Look'N'Feel assigned to our window.*
- virtual **Rect** **getUnclippedInnerRect** () const  
*Get unclipped inner rectangle that our window should return from its member function with the same name.*
- virtual **Rect** **getPixelRect** () const  
*Get actual pixel rectangle our window is to return from its member function with the same name.*
- virtual void **performChildWindowLayout** ()  
*Method called to perform extended laying out of the window's attached child windows.*

## Protected Types

- typedef std::vector< **Property** \* > **PropertyList**

## Protected Member Functions

- void **registerProperty** (**Property** \*property)  
*Register a property class that will be properly managed by this window renderer.*
- virtual void **onAttach** ()  
*Handler called when this windowrenderer is attached to a window.*
- virtual void **onDetach** ()  
*Handler called when this windowrenderer is detached from its window.*
- virtual void **onLookAndFeelAssigned** ()

*Handler called when a Look'N'Feel is assigned to our window.*

- virtual void [onLookNFeelUnassigned \(\)](#)

*Handler called when a Look'N'Feel is removed/unassigned from our window.*

## Protected Attributes

- [Window \\* d\\_window](#)

*Pointer to the window this windowrenderer is assigned to.*

- const [String d\\_name](#)

*Name of the factory type used to create this window renderer.*

- const [String d\\_class](#)

*Name of the widget class that is the "minimum" requirement.*

- [PropertyList d\\_properties](#)

*The list of properties that this windowrenderer will be handling.*

## Friends

- class [Window](#)

### 6.347.1 Detailed Description

Base-class for the assignable [WindowRenderer](#) object.

### 6.347.2 Constructor & Destructor Documentation

#### 6.347.2.1 CEGUI::WindowRenderer::WindowRenderer (const String & *name*, const String & *class\_name* = "Window")

Constructor.

##### Parameters:

*name* Factory type name

*class\_name* The name of a widget class that is to be the minimum requirement for this window renderer.

### 6.347.3 Member Function Documentation

#### 6.347.3.1 virtual void CEGUI::WindowRenderer::render () [pure virtual]

Populate render cache.

This method must be implemented by all window renderers and should perform the rendering operations needed for this widget. Normally using the Falagard API...

### 6.347.3.2 void CEGUI::WindowRenderer::registerProperty (Property \* *property*) [protected]

Register a property class that will be properly managed by this window renderer.

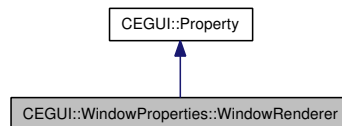
#### Parameters:

*property* Pointer to a static [Property](#) object that will be added to the target window.

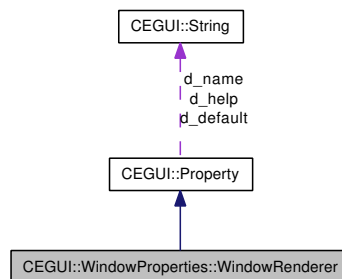
## 6.348 CEGUI::WindowProperties::WindowRenderer Class Reference

[Property](#) to access/change the assigned window renderer object.

Inheritance diagram for CEGUI::WindowProperties::WindowRenderer:



Collaboration diagram for CEGUI::WindowProperties::WindowRenderer:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*
- void [writeXMLToStream](#) (const [PropertyReceiver](#) \*receiver, [XMLSerializer](#) &xml\_stream) const  
*Writes out an XML representation of this class to the given stream.*

### 6.348.1 Detailed Description

[Property](#) to access/change the assigned window renderer object.

**Usage:**

- Name: [WindowRenderer](#)
- Format: "[windowRendererName]"

Where [windowRendererName] is the factory name of the window renderer type you wish to assign.

## 6.348.2 Member Function Documentation

### 6.348.2.1 **String CEGUI::WindowProperties::WindowRenderer::get (const PropertyReceiver \* *receiver*) const** [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.348.2.2 **void CEGUI::WindowProperties::WindowRenderer::set (PropertyReceiver \* *receiver*, const String & *value*)** [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

### 6.348.2.3 **void CEGUI::WindowProperties::WindowRenderer::writeXMLToStream (const PropertyReceiver \* *receiver*, XMLSerializer & *xml\_stream*) const** [virtual]

Writes out an XML representation of this class to the given stream.

#### Note:

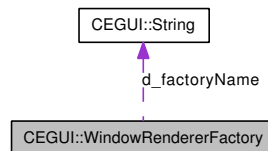
This would normally have been implemented via XMLGenerator base class, but in this case we require the target [PropertyReceiver](#) in order to obtain the property value.

Reimplemented from [CEGUI::Property](#).

## 6.349 CEGUI::WindowRendererFactory Class Reference

Base-class for [WindowRendererFactory](#).

Collaboration diagram for CEGUI::WindowRendererFactory:



### Public Member Functions

- [WindowRendererFactory](#) (const [String](#) &name)  
*Constructor.*
- virtual [~WindowRendererFactory](#) ()  
*Destructor.*
- const [String](#) & [getName](#) () const  
*Returns the type name of this window renderer factory.*
- virtual [WindowRenderer](#) \* [create](#) ()=0  
*Creates and returns a new window renderer object.*
- virtual void [destroy](#) ([WindowRenderer](#) \*wr)=0  
*Destroys a window renderer object previously created by us.*

### Protected Attributes

- [String](#) d\_factoryName  
*Our factory type name.*

#### 6.349.1 Detailed Description

Base-class for [WindowRendererFactory](#).

#### 6.349.2 Constructor & Destructor Documentation

##### 6.349.2.1 CEGUI::WindowRendererFactory::WindowRendererFactory (const String & name) [inline]

Constructor.

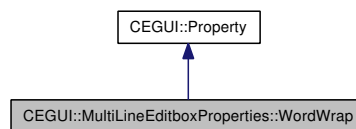
**Parameters:**

*name* Type name for this window renderer factory

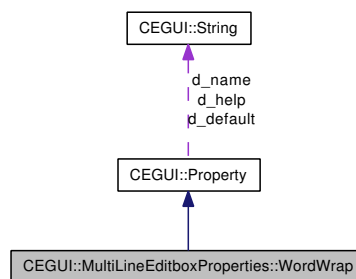
## 6.350 CEGUI::MultiLineEditboxProperties::WordWrap Class Reference

[Property](#) to access the word-wrap setting of the edit box.

Inheritance diagram for CEGUI::MultiLineEditboxProperties::WordWrap:



Collaboration diagram for CEGUI::MultiLineEditboxProperties::WordWrap:



### Public Member Functions

- [String](#) `get` (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void `set` ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.350.1 Detailed Description

[Property](#) to access the word-wrap setting of the edit box.

**Usage:**

- Name: [WordWrap](#)
- Format: "[text]"

**Where [Text] is:**

- "True" to indicate the text should be word-wrapped.
- "False" to indicate the text should not be word-wrapped.



## 6.350.2 Member Function Documentation

### 6.350.2.1 String CEGUI::MultiLineEditboxProperties::WordWrap::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

#### Parameters:

*receiver* Pointer to the target object.

#### Returns:

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.350.2.2 void CEGUI::MultiLineEditboxProperties::WordWrap::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

#### Parameters:

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

#### Returns:

Nothing.

#### Exceptions:

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## 6.351 CEGUI::XMLAttributes Class Reference

Class representing a block of attributes associated with an XML element.

### Public Member Functions

- [XMLAttributes](#) (void)  
*XMLAttributes constructor.*
- virtual [~XMLAttributes](#) (void)  
*XMLAttributes Destructor.*
- void [add](#) (const [String](#) &attrName, const [String](#) &attrValue)  
*Adds an attribute to the attribute block. If the attribute value already exists, it is replaced with the new value.*
- void [remove](#) (const [String](#) &attrName)  
*Removes an attribute from the attribute block.*
- bool [exists](#) (const [String](#) &attrName) const  
*Return whether the named attribute exists within the attribute block.*
- size\_t [getCount](#) (void) const  
*Return the number of attributes in the attribute block.*
- const [String](#) & [getName](#) (size\_t index) const  
*Return the name of an attribute based upon its index within the attribute block.*
- const [String](#) & [getValue](#) (size\_t index) const  
*Return the value string of an attribute based upon its index within the attribute block.*
- const [String](#) & [getValue](#) (const [String](#) &attrName) const  
*Return the value string for attribute attrName.*
- const [String](#) & [getValueAsString](#) (const [String](#) &attrName, const [String](#) &def="") const  
*Return the value of attribute attrName as a string.*
- bool [getValueAsBool](#) (const [String](#) &attrName, bool def=false) const  
*Return the value of attribute attrName as a boolean value.*
- int [getValueAsInteger](#) (const [String](#) &attrName, int def=0) const  
*Return the value of attribute attrName as a integer value.*
- float [getValueAsFloat](#) (const [String](#) &attrName, float def=0.0f) const  
*Return the value of attribute attrName as a floating point value.*

### Protected Types

- typedef std::map< [String](#), [String](#), [String::FastLessCompare](#) > [AttributeMap](#)

## Protected Attributes

- AttributeMap **d\_attrs**

### 6.351.1 Detailed Description

Class representing a block of attributes associated with an XML element.

### 6.351.2 Member Function Documentation

#### 6.351.2.1 void CEGUI::XMLAttributes::add (const String & *attrName*, const String & *attrValue*)

Adds an attribute to the attribute block. If the attribute value already exists, it is replaced with the new value.

##### Parameters:

*attrName* String object holding the name of the attribute to be added.

*attrValue* String object holding a string representation of the attribute value.

##### Returns:

Nothing.

#### 6.351.2.2 void CEGUI::XMLAttributes::remove (const String & *attrName*)

Removes an attribute from the attribute block.

##### Parameters:

*attrName* String object holding the name of the attribute to be removed.

##### Returns:

Nothing.

#### 6.351.2.3 bool CEGUI::XMLAttributes::exists (const String & *attrName*) const

Return whether the named attribute exists within the attribute block.

##### Parameters:

*attrName* String object holding the name of the attribute to be checked.

##### Returns:

- true if an attribute with the name *attrName* is present in the attribute block.
- false if no attribute named *attrName* is present in the attribute block.

**6.351.2.4   size\_t CEGUI::XMLAttributes::getCount (void) const**

Return the number of attributes in the attribute block.

**Returns:**

value specifying the number of attributes in this attribute block.

**6.351.2.5   const String & CEGUI::XMLAttributes::getName (size\_t index) const**

Return the name of an attribute based upon its index within the attribute block.

**Note:**

Nothing is specified about the order of elements within the attribute block. Elements may not, for example, appear in the order they were specified in the XML file.

**Parameters:**

*index* zero based index of the attribute whos name is to be returned.

**Returns:**

[String](#) object holding the name of the attribute at the requested index.

**Exceptions:**

*IllegalRequestException* thrown if *index* is out of range for this attribute block.

**6.351.2.6   const String & CEGUI::XMLAttributes::getValue (size\_t index) const**

Return the value string of an attribute based upon its index within the attribute block.

**Note:**

Nothing is specified about the order of elements within the attribute block. Elements may not, for example, appear in the order they were specified in the XML file.

**Parameters:**

*index* zero based index of the attribute whos value string is to be returned.

**Returns:**

[String](#) object holding the string value of the attribute at the requested index.

**Exceptions:**

*IllegalRequestException* thrown if *index* is out of range for this attribute block.

**6.351.2.7 const String & CEGUI::XMLAttributes::getValue (const String & attrName) const**

Return the value string for attribute *attrName*.

**Parameters:**

*attrName* [String](#) object holding the name of the attribute whos value string is to be returned

**Returns:**

[String](#) object hilding the value string for attribute *attrName*.

**Exceptions:**

[UnknownObjectException](#) thrown if no attribute named *attrName* is present in the attribute block.

**6.351.2.8 const String & CEGUI::XMLAttributes::getValueAsString (const String & attrName, const String & def = "") const**

Return the value of attribute *attrName* as a string.

**Parameters:**

*attrName* [String](#) object holding the name of the attribute whos value is to be returned.

*def* [String](#) object holding the default value to be returned if *attrName* does not exist in the attribute block. For some parsers, defaults can be gotten from schemas and such like, though for others this may not be desired or possible, so this parameter is used to ensure a default is available in the absence of other mechanisms.

**Returns:**

[String](#) object containing the value of attribute *attrName* if present, or *def* if not.

**6.351.2.9 bool CEGUI::XMLAttributes::getValueAsBool (const String & attrName, bool def = false) const**

Return the value of attribute *attrName* as a boolean value.

**Parameters:**

*attrName* [String](#) object holding the name of the attribute whos value is to be returned.

*def* bool value specifying the default value to be returned if *attrName* does not exist in the attribute block. For some parsers, defaults can be gotten from schemas and such like, though for others this may not be desired or possible, so this parameter is used to ensure a default is available in the absence of other mechanisms.

**Returns:**

bool value equal to the value of attribute *attrName* if present, or *def* if not.

**Exceptions:**

[IllegalRequestException](#) thrown if the attribute value string coul dnot be converted to the requested type.

**6.351.2.10** `int CEGUI::XMLAttributes::getValueAsInteger (const String & attrName, int def = 0) const`

Return the value of attribute *attrName* as a integer value.

**Parameters:**

*attrName* [String](#) object holding the name of the attribute whos value is to be returned.

*def* integer value specifying the default value to be returned if *attrName* does not exist in the attribute block. For some parsers, defaults can be gotten from schemas and such like, though for others this may not be desired or possible, so this parameter is used to ensure a default is available in the absence of other mechanisms.

**Returns:**

integer value equal to the value of attribute *attrName* if present, or *def* if not.

**Exceptions:**

*IllegalRequestException* thrown if the attribute value string coul dnot be converted to the requested type.

**6.351.2.11** `float CEGUI::XMLAttributes::getValueAsFloat (const String & attrName, float def = 0.0f) const`

Return the value of attribute *attrName* as a floating point value.

**Parameters:**

*attrName* [String](#) object holding the name of the attribute whos value is to be returned.

*def* float value specifying the default value to be returned if *attrName* does not exist in the attribute block. For some parsers, defaults can be gotten from schemas and such like, though for others this may not be desired or possible, so this parameter is used to ensure a default is available in the absence of other mechanisms.

**Returns:**

float value equal to the value of attribute *attrName* if present, or *def* if not.

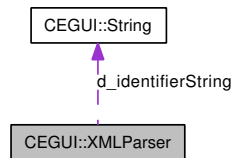
**Exceptions:**

*IllegalRequestException* thrown if the attribute value string coul dnot be converted to the requested type.

## 6.352 CEGUI::XMLParser Class Reference

This is an abstract class that is used by [CEGUI](#) to interface with XML parser libraries.

Collaboration diagram for CEGUI::XMLParser:



### Public Member Functions

- [XMLParser](#) (void)  
*XMLParser base class constructor.*
- virtual [~XMLParser](#) (void)  
*XMLParser base class destructor.*
- bool [initialise](#) (void)  
*Initialises the XMLParser module ready for use.*
- void [cleanup](#) (void)  
*Cleans up the XMLParser module after use.*
- virtual void [parseXMLFile](#) (XMLHandler &handler, const [String](#) &filename, const [String](#) &schemaname, const [String](#) &resourceGroup)=0  
*abstract method which initiates parsing of an XML file.*
- const [String](#) & [getIdentifierString](#) () const  
*Return identification string for the XML parser module. If the internal id string has not been set by the XML parser module creator, a generic string of "Unknown XML parser" will be returned.*

### Protected Member Functions

- virtual bool [initialiseImpl](#) (void)=0  
*abstract method which initialises the XMLParser ready for use.*
- virtual void [cleanupImpl](#) (void)=0  
*abstract method which cleans up the XMLParser after use.*

### Protected Attributes

- [String d\\_identifierString](#)  
*String that holds some id information about the module.*

### 6.352.1 Detailed Description

This is an abstract class that is used by [CEGUI](#) to interface with XML parser libraries.

### 6.352.2 Member Function Documentation

#### 6.352.2.1 `bool CEGUI::XMLParser::initialise (void)`

Initialises the [XMLParser](#) module ready for use.

Note that this calls the protected abstract method 'initialiseImpl', which should be provided in your implementation to perform any required initialisation.

##### Returns:

- true if the module initialised successfully.
- false if the module initialisation failed.

#### 6.352.2.2 `void CEGUI::XMLParser::cleanup (void)`

Cleans up the [XMLParser](#) module after use.

Note that this calls the protected abstract method 'cleanupImpl', which should be provided in your implementation to perform any required cleanup.

##### Returns:

Nothing.

#### 6.352.2.3 `virtual void CEGUI::XMLParser::parseXMLFile (XMLHandler & handler, const String & filename, const String & schemaName, const String & resourceGroup)` [pure virtual]

abstract method which initiates parsing of an XML file.

##### Parameters:

*handler* XMLHandler based object which will process the XML elements.

*filename* [String](#) object holding the filename of the XML file to be parsed.

*schemaName* [String](#) object holding the name of the XML schema file to use for validating the XML.  
Note that whether this is used or not is dependant upon the [XMLParser](#) in use.

*resourceGroup* [String](#) object holding the resource group identifier which will be passed to the [ResourceProvider](#) when loading the XML and schema files.

##### Returns:

Nothing.



**6.352.2.4 const String & CEGUI::XMLParser::getIdentifierString () const**

Return identification string for the XML parser module. If the internal id string has not been set by the XML parser module creator, a generic string of "Unknown XML parser" will be returned.

**Returns:**

[String](#) object holding a string that identifies the XML parser in use.

**6.352.2.5 virtual bool CEGUI::XMLParser::initialiseImpl (void) [protected, pure virtual]**

abstract method which initialises the [XMLParser](#) ready for use.

**Returns:**

- true if the module initialised successfully.
- false if the module initialisation failed.

**6.352.2.6 virtual void CEGUI::XMLParser::cleanupImpl (void) [protected, pure virtual]**

abstract method which cleans up the [XMLParser](#) after use.

**Returns:**

Nothing.

## 6.353 CEGUI::XMLSerializer Class Reference

Class used to create XML Document.

### Public Member Functions

- [XMLSerializer](#) ([OutStream](#) &out, size\_t indentSpace=4)  
*XMLSerializer constructor.*
- virtual [~XMLSerializer](#) (void)  
*XMLSerializer destructor.*
- [XMLSerializer](#) & [openTag](#) (const [String](#) &name)  
*Start a new tag in the xml document.*
- [XMLSerializer](#) & [closeTag](#) (void)  
*Close the current tag.*
- [XMLSerializer](#) & [attribute](#) (const [String](#) &name, const [String](#) &value)  
*After an opening tag you can populate attribute list with this function.*
- [XMLSerializer](#) & [text](#) (const [String](#) &text)  
*Create a text node.*
- unsigned int [getTagCount](#) () const  
*report the number of tags created in the document*
- [operator bool](#) () const  
*Check whether the XML Serializer status is valid.*
- bool [operator!](#) () const  
*Check whether the XML Serializer status is invalid.*

### 6.353.1 Detailed Description

Class used to create XML Document.

This class hides the complexity of formatting valid XML files. The class provides automatic substitution of entities, XML indenting in respect of the spaces. It does not contain any codes specific to [CEGUI](#) taking apart the [CEGUI::String](#) class. The following example shows the code needed to export parts of an XML document similar to what can be found in a layout.

```
#include <iostream>
#include <CEGUI/XMLSerializer.h>

int main()
{
    Create an encoder that outputs its result on standard output
    XMLSerializer xml(std::cout, 4);
    xml.openTag("Window")
        .attribute("Type", "TaharezLook/StaticText")
```

```
.attribute("Name", "Test")
.openTag("Property")
.attribute("Name", "Text")
.text("This is the static text to be displayed")
.closeTag()
.openTag("Window")
.attribute("Name", "Button")
.attribute("Type", "Vanilla/Button")
.openTag("Property")
.attribute("Name", "Text")
.attribute("Value", "Push me")
.closeTag()
.closeTag()
.closeTag();

if (xml)
{
    std::cout << "XML Exported successfully" << std::endl;
}
return 0;
}
```

## 6.353.2 Constructor & Destructor Documentation

### 6.353.2.1 CEGUI::XMLSerializer::XMLSerializer (OutputStream & *out*, size\_t *indentSpace* = 4)

[XMLSerializer](#) constructor.

#### Parameters:

- out* The stream to use to export the result  
*indentSpace* The indentation level (0 to disable indentation)

## 6.353.3 Member Function Documentation

### 6.353.3.1 XMLSerializer & CEGUI::XMLSerializer::openTag (const String & *name*)

Start a new tag in the xml document.

#### Parameters:

- name* The name of the tag

#### Returns:

- A reference to the current object for chaining operation

### 6.353.3.2 XMLSerializer & CEGUI::XMLSerializer::closeTag (void)

Close the current tag.

#### Returns:

- A reference to the current object for chaining operation

**6.353.3.3 XMLSerializer & CEGUI::XMLSerializer::attribute (const String & *name*, const String & *value*)**

After an opening tag you can populate attribute list with this function.

**Parameters:**

*name* The name of the attribute

*value* The value of the attribute

**Returns:**

A reference to the current object for chaining operation

**6.353.3.4 XMLSerializer & CEGUI::XMLSerializer::text (const String & *text*)**

Create a text node.

**Parameters:**

*text* the content of the node

**Returns:**

A reference to the current object for chaining operation

**6.353.3.5 unsigned int CEGUI::XMLSerializer::getTagCount () const**

report the number of tags created in the document

**Returns:**

return the number of tag created in the document

**6.353.3.6 CEGUI::XMLSerializer::operator bool () const [inline]**

Check whether the XML Serializer status is valid.

**Returns:**

True if all previous operations were successful

**6.353.3.7 bool CEGUI::XMLSerializer::operator! () const [inline]**

Check whether the XML Serializer status is invalid.

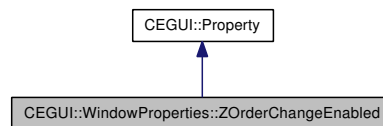
**Returns:**

True if one operation failed

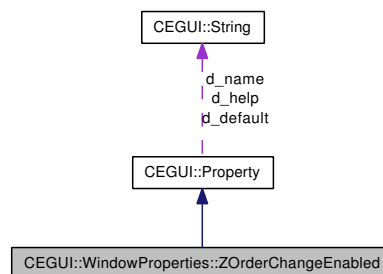
## 6.354 CEGUI::WindowProperties::ZOrderChangeEnabled Class Reference

[Property](#) to access window Z-Order changing enabled setting.

Inheritance diagram for CEGUI::WindowProperties::ZOrderChangeEnabled:



Collaboration diagram for CEGUI::WindowProperties::ZOrderChangeEnabled:



### Public Member Functions

- [String](#) [get](#) (const [PropertyReceiver](#) \*receiver) const  
*Return the current value of the [Property](#) as a [String](#).*
- void [set](#) ([PropertyReceiver](#) \*receiver, const [String](#) &value)  
*Sets the value of the property.*

#### 6.354.1 Detailed Description

[Property](#) to access window Z-Order changing enabled setting.

This property offers access to the setting that controls whether z-order changes are enabled for the window.

Usage:

- Name: [ZOrderChangeEnabled](#)
- Format: "[text]".

Where [Text] is:

- "True" to indicate the [Window](#) should respect requests to change z-order.
- "False" to indicate the [Window](#) should not change it's z-order.

## 6.354.2 Member Function Documentation

### 6.354.2.1 String CEGUI::WindowProperties::ZOrderChangeEnabled::get (const PropertyReceiver \* *receiver*) const [virtual]

Return the current value of the [Property](#) as a [String](#).

**Parameters:**

*receiver* Pointer to the target object.

**Returns:**

[String](#) object containing a textual representation of the current value of the [Property](#)

Implements [CEGUI::Property](#).

### 6.354.2.2 void CEGUI::WindowProperties::ZOrderChangeEnabled::set (PropertyReceiver \* *receiver*, const String & *value*) [virtual]

Sets the value of the property.

**Parameters:**

*receiver* Pointer to the target object.

*value* A [String](#) object that contains a textual representation of the new value to assign to the [Property](#).

**Returns:**

Nothing.

**Exceptions:**

[InvalidRequestException](#) Thrown when the [Property](#) was unable to interpret the content of *value*.

Implements [CEGUI::Property](#).

## Chapter 7

# Crazy Eddies GUI System Page Documentation

### 7.1 Todo List

**Class [CEGUI::ItemEntry](#)** Fire events on selection / deselection. (Maybe selectable mode changed as well?)

**Class [CEGUI::ScriptWindowHelper](#)**

**Class [CEGUI::WidgetComponent](#)** This is not finished in the slightest! There will be many changes here...

**Class [CEGUI::WindowFactoryManager](#)** I think we could clean up the mapping stuff a bit. Possibly make it more generic now with the window renderers and all.

## 7.2 Bug List

Class [CEGUI::ScriptWindowHelper](#)



# Index

- ~DynamicModule
  - CEGUI::DynamicModule, [252](#)
- ~FactoryModule
  - CEGUI::FactoryModule, [290](#)
- ~Scheme
  - CEGUI::Scheme, [815](#)
- ~WindowManager
  - CEGUI::WindowManager, [1312](#)
- AbsoluteDim
  - CEGUI::AbsoluteDim, [84](#)
- activate
  - CEGUI::Window, [1264](#)
- activateEditbox
  - CEGUI::Combobox, [166](#)
- Active
  - CEGUI::Tooltip, [1117](#)
- add
  - CEGUI::XMLAttributes, [1329](#)
- addChildWindow
  - CEGUI::Window, [1266](#)
- addColumn
  - CEGUI::ListHeader, [565](#)
  - CEGUI::MultiColumnList, [660](#)
- addEvent
  - CEGUI::EventSet, [281](#)
- addFactory
  - CEGUI::WindowFactoryManager, [1303](#)
- addFalagardWindowMapping
  - CEGUI::WindowFactoryManager, [1305](#)
- addFrameComponent
  - CEGUI::ImagerySection, [451](#)
- addImageryComponent
  - CEGUI::ImagerySection, [450](#)
- addImagerySection
  - CEGUI::WidgetLookFeel, [1210](#)
- addItem
  - CEGUI::Combobox, [167](#)
  - CEGUI::ItemListBase, [496](#)
  - CEGUI::Listbox, [529](#)
  - CEGUI::Tree, [1135](#)
- addLayer
  - CEGUI::StateImagery, [960](#)
- addNamedArea
  - CEGUI::WidgetLookFeel, [1213](#)
- addProperty
  - CEGUI::PropertySet, [764](#)
- addPropertyDefinition
  - CEGUI::WidgetLookFeel, [1214](#)
- addPropertyInitialiser
  - CEGUI::WidgetLookFeel, [1211](#)
- addPropertyLinkDefinition
  - CEGUI::WidgetLookFeel, [1214](#)
- addQuad
  - CEGUI::Renderer, [793](#)
- addRow
  - CEGUI::MultiColumnList, [662](#)
- addSectionSpecification
  - CEGUI::LayerSpecification, [519](#)
- addStateSpecification
  - CEGUI::WidgetLookFeel, [1211](#)
- addTab
  - CEGUI::TabControl, [1069](#)
- addTextComponent
  - CEGUI::ImagerySection, [451](#)
- addWidgetComponent
  - CEGUI::WidgetLookFeel, [1211](#)
- addWidgetLook
  - CEGUI::WidgetLookManager, [1219](#)
- addWindowToDrawList
  - CEGUI::Window, [1296](#)
- addWindowTypeAlias
  - CEGUI::WindowFactoryManager, [1304](#)
- adjustProgress
  - CEGUI::ProgressBar, [737](#)
- advanceZValue
  - CEGUI::Renderer, [797](#)
- AlreadyExistsException
  - CEGUI::AlreadyExistsException, [92](#)
- Alt
  - CEGUI, [49](#)
- append
  - CEGUI::String, [995–1000](#)
- apply
  - CEGUI::PropertyInitialiser, [758](#)
- Ascending
  - CEGUI::ListHeaderSegment, [576](#)
- assign
  - CEGUI::String, [991–995](#)
- at

- CEGUI::String, 989
- attribute
  - CEGUI::XMLSerializer, 1337
- autoSizeColumnHeader
  - CEGUI::MultiColumnList, 670
- begin
  - CEGUI::String, 1033
- BlankMouseCursor
  - CEGUI, 48
- BottomLeftToTopRight
  - CEGUI, 49
- BoundSlot
  - CEGUI::BoundSlot, 107
- c\_str
  - CEGUI::String, 989
- cacheImage
  - CEGUI::RenderCache, 789
- cacheText
  - CEGUI::RenderCache, 789
- cacheTreeBaseImagery
  - CEGUI::Tree, 1139
- calculateTabButtonSizePosition
  - CEGUI::TabControl, 1072
- capacity
  - CEGUI::String, 982
- captureInput
  - CEGUI::Window, 1268
- CEGUI, 25
  - Alt, 49
  - BlankMouseCursor, 48
  - BottomLeftToTopRight, 49
  - Centred, 49
  - Control, 49
  - DefaultMouseCursor, 48
  - DimensionOperator, 46
  - DimensionType, 46
  - DOP\_ADD, 46
  - DOP\_DIVIDE, 46
  - DOP\_MULTIPLY, 46
  - DOP\_NOOP, 46
  - DOP\_SUBTRACT, 46
  - DT\_BOTTOM\_EDGE, 46
  - DT\_HEIGHT, 46
  - DT\_INVALID, 46
  - DT\_LEFT\_EDGE, 46
  - DT\_RIGHT\_EDGE, 46
  - DT\_TOP\_EDGE, 46
  - DT\_WIDTH, 46
  - DT\_X\_OFFSET, 46
  - DT\_X\_POSITION, 46
  - DT\_Y\_OFFSET, 46
  - DT\_Y\_POSITION, 46
  - Errors, 48
  - FIC\_BACKGROUND, 47
  - FIC\_BOTTOM\_EDGE, 47
  - FIC\_BOTTOM\_LEFT\_CORNER, 47
  - FIC\_BOTTOM\_RIGHT\_CORNER, 47
  - FIC\_FRAME\_IMAGE\_COUNT, 47
  - FIC\_LEFT\_EDGE, 47
  - FIC\_RIGHT\_EDGE, 47
  - FIC\_TOP\_EDGE, 47
  - FIC\_TOP\_LEFT\_CORNER, 47
  - FIC\_TOP\_RIGHT\_CORNER, 47
  - FlipHorizontal, 49
  - FlipVertical, 49
  - FMT\_BASELINE, 47
  - FMT\_HORZ\_EXTENT, 47
  - FMT\_LINE\_SPACING, 47
  - FontMetricType, 46
  - FrameImageComponent, 47
  - HA\_CENTRE, 47
  - HA\_LEFT, 47
  - HA\_RIGHT, 47
  - HF\_CENTRE\_ALIGNED, 47
  - HF\_LEFT\_ALIGNED, 47
  - HF\_RIGHT\_ALIGNED, 47
  - HF\_STRETCHED, 47
  - HF\_TILED, 47
  - HorizontalAlignment, 47
  - HorizontalFormatting, 47
  - HorizontalTextFormatting, 47
  - HTF\_CENTRE\_ALIGNED, 48
  - HTF\_JUSTIFIED, 48
  - HTF\_LEFT\_ALIGNED, 48
  - HTF\_RIGHT\_ALIGNED, 48
  - HTF\_WORDWRAP\_CENTRE\_ALIGNED, 48
  - HTF\_WORDWRAP\_JUSTIFIED, 48
  - HTF\_WORDWRAP\_LEFT\_ALIGNED, 48
  - HTF\_WORDWRAP\_RIGHT\_ALIGNED, 48
  - Informative, 48
  - Insane, 48
  - Justified, 49
  - LeftAligned, 49
  - LeftMouse, 49
  - LoggingLevel, 48
  - MiddleMouse, 49
  - MouseButton, 48
  - MouseCursorImage, 48
  - NoButton, 48
  - operator+, 51–53
  - OrientationFlags, 48
  - QuadSplitMode, 49
  - RightAligned, 49
  - RightMouse, 49
  - RotateRightAngle, 49

- Shift, [49](#)
- Standard, [48](#)
- swap, [53](#)
- SystemKey, [49](#)
- TextFormatting, [49](#)
- TopLeftToBottomRight, [49](#)
- VA\_BOTTOM, [50](#)
- VA\_CENTRE, [50](#)
- VA\_TOP, [50](#)
- VerticalAlignment, [50](#)
- VerticalFormatting, [50](#)
- VerticalTextFormatting, [50](#)
- VF\_BOTTOM\_ALIGNED, [50](#)
- VF\_CENTRE\_ALIGNED, [50](#)
- VF\_STRETCHED, [50](#)
- VF\_TILED, [50](#)
- VF\_TOP\_ALIGNED, [50](#)
- VTF\_BOTTOM\_ALIGNED, [50](#)
- VTF\_CENTRE\_ALIGNED, [50](#)
- VTF\_TOP\_ALIGNED, [50](#)
- Warnings, [48](#)
- WordWrapCentred, [50](#)
- WordWrapJustified, [50](#)
- WordWrapLeftAligned, [50](#)
- WordWrapRightAligned, [50](#)
- X1Mouse, [49](#)
- X2Mouse, [49](#)
- CEGUI::AbsoluteDim, [83](#)
  - AbsoluteDim, [84](#)
- CEGUI::ActivationEventArgs, [85](#)
- CEGUI::AlreadyExistsException, [92](#)
  - AlreadyExistsException, [92](#)
- CEGUI::BaseDim, [102](#)
  - clone, [103](#)
  - getDimensionOperator, [104](#)
  - getOperand, [104](#)
  - getValue, [103](#)
  - setDimensionOperator, [104](#)
  - setOperand, [104](#)
  - writeXMLToStream, [104](#)
- CEGUI::BoundSlot, [106](#)
  - BoundSlot, [107](#)
  - connected, [107](#)
  - disconnect, [107](#)
  - operator!=, [108](#)
  - operator==, [107](#)
- CEGUI::ButtonBase, [109](#)
  - isHovering, [110](#)
  - isPushed, [110](#)
  - onCaptureLost, [111](#)
  - onMouseButtonDown, [110](#)
  - onMouseButtonUp, [111](#)
  - onMouseLeaves, [111](#)
  - onMouseMove, [110](#)
  - testClassName\_impl, [112](#)
  - updateInternalState, [111](#)
- CEGUI::Checkbox, [119](#)
  - isSelected, [120](#)
  - onMouseButtonUp, [120](#)
  - setSelected, [120](#)
  - testClassName\_impl, [121](#)
- CEGUI::CheckboxProperties, [55](#)
- CEGUI::CheckboxProperties::Selected, [885](#)
  - get, [886](#)
  - set, [886](#)
- CEGUI::ClippedContainer, [130](#)
  - drawSelf, [134](#)
  - getClipArea, [133](#)
  - getUnclippedInnerRect\_impl, [133](#)
  - setClipperWindow, [133](#)
  - testClassName\_impl, [133](#)
- CEGUI::colour, [137](#)
- CEGUI::ColourRect, [138](#)
  - getColourAtPoint, [141](#)
  - getSubRectangle, [140](#)
  - isMonochromatic, [140](#)
  - modulateAlpha, [141](#)
  - setAlpha, [139](#)
  - setBottomAlpha, [139](#)
  - setColours, [141](#)
  - setLeftAlpha, [140](#)
  - setRightAlpha, [140](#)
  - setTopAlpha, [139](#)
- CEGUI::Combobox, [152](#)
  - activateEditbox, [166](#)
  - addItem, [167](#)
  - clearAllSelections, [167](#)
  - findItemWithText, [163](#)
  - getCaratIndex, [161](#)
  - getDropList, [160](#)
  - getEditbox, [159](#)
  - getItemCount, [162](#)
  - getItemIndex, [162](#)
  - getListboxItemFromIndex, [162](#)
  - getMaxTextLength, [161](#)
  - getPushButton, [159](#)
  - getSelectedItem, [162](#)
  - getSelectionEndIndex, [161](#)
  - getSelectionLength, [161](#)
  - getSelectionStartIndex, [161](#)
  - getSingleClickEnabled, [159](#)
  - getValidationString, [161](#)
  - handleUpdatedListItemData, [169](#)
  - hasInputFocus, [160](#)
  - hideDropList, [164](#)
  - initialiseComponents, [164](#)
  - insertItem, [167](#)
  - isDropDownListVisible, [159](#)

- isHit, 159
- isHorzScrollbarAlwaysShown, 164
- isSelected, 163
- isListBoxItemInList, 163
- isReadOnly, 160
- isSortEnabled, 163
- isTextValid, 160
- isVertScrollbarAlwaysShown, 164
- onActivated, 170
- onFontChanged, 169
- onTextChanged, 170
- removeItem, 167
- resetList, 166
- setCaratIndex, 165
- setItemSelectState, 168, 169
- setMaxTextLength, 166
- setReadOnly, 165
- setSelection, 166
- setShowHorzScrollbar, 168
- setShowVertScrollbar, 168
- setSingleClickEnabled, 165
- setSortingEnabled, 167
- setValidationString, 165
- showDropList, 164
- testClassName\_impl, 169
- CEGUI::ComboboxProperties, 56
- CEGUI::ComboboxProperties::CaratIndex, 115
  - get, 116
  - set, 116
- CEGUI::ComboboxProperties::EditSelectionLength, 270
  - get, 271
  - set, 271
- CEGUI::ComboboxProperties::EditSelectionStart, 272
  - get, 273
  - set, 273
- CEGUI::ComboboxProperties::ForceHorzScrollbar, 340
  - get, 341
  - set, 341
- CEGUI::ComboboxProperties::ForceVertScrollbar, 354
  - get, 355
  - set, 355
- CEGUI::ComboboxProperties::MaxEditTextLength, 593
  - get, 594
  - set, 594
- CEGUI::ComboboxProperties::ReadOnly, 778
  - get, 779
  - set, 779
- CEGUI::ComboboxProperties::SingleClickMode, 902
  - get, 903
  - set, 903
- CEGUI::ComboboxProperties::SortList, 940
  - get, 941
  - set, 941
- CEGUI::ComboboxProperties::ValidationString, 1182
  - get, 1183
  - set, 1183
- CEGUI::ComboDropList, 171
  - initialiseComponents, 173
  - isArmed, 174
  - isAutoArmEnabled, 174
  - onActivated, 176
  - onCaptureLost, 175
  - onMouseButtonDown, 175
  - onMouseButtonUp, 175
  - onMouseMove, 175
  - setArmed, 173
  - setAutoArmEnabled, 174
  - testClassName\_impl, 174
- CEGUI::ComponentArea, 177
  - getAreaPropertySource, 179
  - getPixelRect, 178
  - isAreaFetchedFromProperty, 178
  - setAreaPropertySource, 179
  - writeXMLToStream, 178
- CEGUI::Config\_xmlHandler, 180
- CEGUI::ConstBaseIterator, 182
  - ConstBaseIterator, 183
  - operator++, 183
  - operator--, 184
- CEGUI::CoordConverter, 193
  - screenToWindow, 197, 198
  - screenToWindowX, 196
  - screenToWindowY, 196, 197
  - windowToScreen, 195, 196
  - windowToScreenX, 194
  - windowToScreenY, 194, 195
- CEGUI::DefaultLogger, 207
  - logEvent, 208
  - setLogFilename, 208
- CEGUI::Dimension, 211
  - Dimension, 212
  - getBaseDimension, 212
  - getDimensionType, 212
  - setBaseDimension, 212
  - setDimensionType, 212
  - writeXMLToStream, 213
- CEGUI::DragContainer, 226
  - doDragging, 233
  - getCurrentDropTarget, 232
  - getDragAlpha, 231
  - getDragCursorImage, 231

- getPixelDragThreshold, 230
- initialiseDragging, 233
- isBeingDragged, 230
- isDraggingEnabled, 230
- isDraggingThresholdExceeded, 233
- onAlphaChanged, 235
- onCaptureLost, 234
- onClippingChanged, 235
- onDragAlphaChanged, 236
- onDragDropTargetChanged, 237
- onDragEnabledChanged, 236
- onDragEnded, 236
- onDragMouseCursorChanged, 237
- onDragPositionChanged, 236
- onDragStarted, 235
- onDragThresholdChanged, 237
- onMouseDown, 234
- onMouseButtonDown, 234
- onMouseButtonUp, 234
- onMouseMove, 234
- onMoved, 235
- setDragAlpha, 231
- setDragCursorImage, 232
- setDraggingEnabled, 230
- setPixelDragThreshold, 231
- testClassName\_impl, 233
- CEGUI::DragContainerProperties::DragAlpha, 224
  - get, 225
  - set, 225
- CEGUI::DragContainerProperties::DragCursorImage, 239
  - get, 240
  - set, 240
- CEGUI::DragContainerProperties::DraggingEnabled, 246
  - get, 247
  - set, 247
- CEGUI::DragContainerProperties::DragThreshold, 250
  - get, 251
  - set, 251
- CEGUI::DragDropEventArgs, 241
- CEGUI::DynamicModule, 252
  - ~DynamicModule, 252
  - DynamicModule, 252
  - getSymbolAddress, 253
- CEGUI::Editbox, 254
  - clearSelection, 263
  - eraseSelectedText, 263
  - getCaratIndex, 260
  - getMaskCodePoint, 261
  - getMaxTextLength, 261
  - getSelectionEndIndex, 260
  - getSelectionLength, 261
  - getSelectionStartIndex, 260
  - getTextIndexFromPosition, 263
  - getValidationString, 260
  - hasInputFocus, 259
  - isReadOnly, 259
  - isTextMasked, 259
  - isTextValid, 260
  - onCaptureLost, 265
  - onCharacter, 265
  - onKeyDown, 266
  - onMouseButtonDown, 264
  - onMouseButtonUp, 264
  - onMouseDoubleClicked, 265
  - onMouseMove, 265
  - onMouseTripleClicked, 265
  - onTextChanged, 266
  - onTextInvalidatedEvent, 264
  - setCaratIndex, 262
  - setMaskCodePoint, 262
  - setMaxTextLength, 263
  - setReadOnly, 261
  - setSelection, 262
  - setTextMasked, 261
  - setValidationString, 262
  - testClassName\_impl, 264
  - validateWindowRenderer, 264
- CEGUI::EditboxProperties, 57
- CEGUI::EditboxProperties::ActiveSelectionColour, 86
  - get, 87
  - set, 87
- CEGUI::EditboxProperties::CaratIndex, 117
  - get, 118
  - set, 118
- CEGUI::EditboxProperties::InactiveSelectionColour, 472
  - get, 473
  - set, 473
- CEGUI::EditboxProperties::MaskCodepoint, 589
  - get, 590
  - set, 590
- CEGUI::EditboxProperties::MaskText, 591
  - get, 592
  - set, 592
- CEGUI::EditboxProperties::MaxTextLength, 601
  - get, 602
  - set, 602
- CEGUI::EditboxProperties::NormalTextColour, 711
  - get, 712
  - set, 712
- CEGUI::EditboxProperties::ReadOnly, 776
  - get, 777
  - set, 777

- CEGUI::EditboxProperties::SelectedTextColour, 887
  - get, 888
  - set, 888
- CEGUI::EditboxProperties::SelectionLength, 893
  - get, 894
  - set, 894
- CEGUI::EditboxProperties::SelectionStart, 897
  - get, 898
  - set, 898
- CEGUI::EditboxProperties::ValidationString, 1184
  - get, 1185
  - set, 1185
- CEGUI::EditboxWindowRenderer, 267
  - getTextIndexFromPosition, 269
- CEGUI::Event, 274
  - getName, 275
  - operator(), 276
  - subscribe, 275
- CEGUI::Event::ScopedConnection, 277
- CEGUI::EventArgs, 278
- CEGUI::EventSet, 279
  - addEvent, 281
  - fireEvent, 283
  - getEventObject, 284
  - isEventPresent, 281
  - isMuted, 283
  - removeAllEvents, 281
  - removeEvent, 281
  - setMutedState, 283
  - subscribeEvent, 281, 282
  - subscribeScriptedEvent, 282
- CEGUI::Exception, 287
  - Exception, 288
  - getFileName, 288
  - getLine, 289
  - getMessage, 288
  - getName, 288
- CEGUI::FactoryModule, 290
  - ~FactoryModule, 290
  - FactoryModule, 290
  - registerAllFactories, 291
  - registerFactory, 291
- CEGUI::Falagard\_xmlHandler, 298
- CEGUI::FalagardComponentBase, 299
  - getColours, 302
  - getComponentArea, 302
  - initColoursRect, 303
  - render, 301
  - setColours, 302
  - setColoursPropertyIsColourRect, 303
  - setColoursPropertySource, 302
  - setComponentArea, 302
  - setHorzFormattingPropertySource, 303
  - setVertFormattingPropertySource, 303
  - writeColoursXML, 304
  - writeHorzFormatXML, 304
  - writeVertFormatXML, 304
- CEGUI::FalagardXMLHelper, 306
- CEGUI::FileIOException, 307
  - FileIOException, 307
- CEGUI::Font, 311
  - d\_glyphPageLoaded, 325
  - drawText, 318–321
  - Font, 316
  - getBaseline, 322
  - getCharAtPixel, 323
  - getDefaultResourceGroup, 324
  - getFontHeight, 322
  - getFormattedLineCount, 323
  - getFormattedTextExtent, 324
  - getGlyphData, 316
  - getLineSpacing, 322
  - getTextExtent, 322
  - isCodepointAvailable, 317
  - notifyScreenResolution, 321
  - rasterize, 317
  - setDefaultResourceGroup, 324
  - setNativeResolution, 321
  - writeXMLToStream, 317
  - writeXMLToStream\_impl, 317
- CEGUI::Font\_xmlHandler, 326
  - Font\_xmlHandler, 326
- CEGUI::FontDim, 327
  - FontDim, 328
- CEGUI::FontGlyph, 329
  - getAdvance, 330
  - getRenderedAdvance, 330
- CEGUI::FontManager, 331
  - createFont, 332, 333
  - destroyAllFonts, 333
  - destroyFont, 333
  - getFont, 334
  - isFontPresent, 334
  - notifyScreenResolution, 334
  - writeFontToStream, 334
- CEGUI::FontProperties, 58
- CEGUI::FrameComponent, 362
  - getBackgroundHorizontalFormatting, 364
  - getBackgroundVerticalFormatting, 363
  - getImage, 364
  - setBackgroundHorizontalFormatting, 364
  - setBackgroundVerticalFormatting, 363
  - setImage, 364, 365
  - writeXMLToStream, 365
- CEGUI::FrameWindow, 368
  - getCloseButton, 381
  - getEWSizingCursorImage, 377

- getNESWSizingCursorImage, 378
- getNSSizingCursorImage, 377
- getNWSEizingCursorImage, 378
- getSizingBorderAtPoint, 382
- getSizingBorderThickness, 375
- getTitlebar, 381
- initialiseComponents, 374
- isBottomSizingLocation, 383
- isCloseButtonEnabled, 375
- isDragMovingEnabled, 377
- isFrameEnabled, 374
- isHit, 380
- isLeftSizingLocation, 382
- isRightSizingLocation, 382
- isRolledup, 375
- isRollupEnabled, 375
- isSizingEnabled, 374
- isTitleBarEnabled, 374
- isTopSizingLocation, 383
- moveBottomEdge, 382
- moveLeftEdge, 381
- moveRightEdge, 381
- moveTopEdge, 381
- offsetPixelPosition, 377
- onActivated, 384
- onCaptureLost, 384
- onDeactivated, 385
- onMouseButtonDown, 384
- onMouseButtonUp, 384
- onMouseMove, 383
- onTextChanged, 384
- setCloseButtonEnabled, 376
- setDragMovingEnabled, 377
- setEWSizingCursorImage, 378, 379
- setFrameEnabled, 375
- setNESWSizingCursorImage, 379, 380
- setNSSizingCursorImage, 378, 379
- setNWSEizingCursorImage, 378, 380
- setRollupEnabled, 376
- setSizingBorderThickness, 376
- setSizingEnabled, 375
- setTitleBarEnabled, 376
- SizingBottom, 374
- SizingBottomLeft, 374
- SizingBottomRight, 374
- SizingLeft, 374
- SizingLocation, 373
- SizingNone, 374
- SizingRight, 374
- SizingTop, 374
- SizingTopLeft, 374
- SizingTopRight, 374
- testClassName\_impl, 383
- toggleRollup, 376
- CEGUI::FrameWindowProperties, 59
- CEGUI::FrameWindowProperties::CloseButtonEnabled, 135
  - get, 136
  - set, 136
- CEGUI::FrameWindowProperties::DragMovingEnabled, 248
  - get, 249
  - set, 249
- CEGUI::FrameWindowProperties::EWSizingCursorImage, 285
  - get, 285
  - set, 286
- CEGUI::FrameWindowProperties::FrameEnabled, 366
  - get, 367
  - set, 367
- CEGUI::FrameWindowProperties::NESWSizingCursorImage, 705
  - get, 705
  - set, 706
- CEGUI::FrameWindowProperties::NSSizingCursorImage, 713
  - get, 713
  - set, 714
- CEGUI::FrameWindowProperties::NWSEizingCursorImage, 717
  - get, 717
  - set, 718
- CEGUI::FrameWindowProperties::RollUpEnabled, 808
  - get, 809
  - set, 809
- CEGUI::FrameWindowProperties::RollUpState, 810
  - get, 811
  - set, 811
- CEGUI::FrameWindowProperties::SizingBorderThickness, 907
  - get, 908
  - set, 908
- CEGUI::FrameWindowProperties::SizingEnabled, 911
  - get, 912
  - set, 912
- CEGUI::FrameWindowProperties::TitlebarEnabled, 1110
  - get, 1111
  - set, 1111
- CEGUI::FreeFunctionSlot, 386
- CEGUI::FreeTypeFont, 387
  - drawGlyphToBuffer, 389
  - getTextureSize, 389
- CEGUI::FunctorCopySlot, 391



- CEGUI::FunctorPointerSlot, 392
- CEGUI::FunctorReferenceBinder, 393
- CEGUI::FunctorReferenceSlot, 394
- CEGUI::GenericException, 395
  - GenericException, 395
- CEGUI::GlobalEventSet, 397
  - fireEvent, 398
  - getSingleton, 397
  - getSingletonPtr, 398
- CEGUI::GroupBox, 399
  - getContentPane, 402
  - testClassName\_impl, 402
- CEGUI::GUILayout\_xmlHandler, 405
- CEGUI::GUISheet, 408
  - testClassName\_impl, 410
- CEGUI::HeaderSequenceEventArgs, 411
- CEGUI::Image, 430
  - draw, 434–437
  - getHeight, 432
  - getImageset, 433
  - getImagesetName, 433
  - getName, 433
  - getOffsets, 433
  - getOffsetX, 433
  - getOffsetY, 433
  - getSize, 432
  - getSourceTextureArea, 434
  - getWidth, 432
  - Image, 432
  - writeXMLToStream, 437
- CEGUI::ImageCodec, 439
  - getIdentifierString, 439
  - getSupportedFormat, 440
  - load, 440
- CEGUI::ImageDim, 441
  - ImageDim, 442
  - setSourceDimension, 442
  - setSourceImage, 442
- CEGUI::ImageryComponent, 443
  - getHorizontalFormatting, 446
  - getImage, 445
  - getImagePropertySource, 446
  - getVerticalFormatting, 445
  - isImageFetchedFromProperty, 446
  - setHorizontalFormatting, 446
  - setImage, 445
  - setImagePropertySource, 447
  - setVerticalFormatting, 445
  - writeXMLToStream, 446
- CEGUI::ImagerySection, 448
  - addFrameComponent, 451
  - addImageryComponent, 450
  - addTextComponent, 451
  - clearFrameComponents, 451
  - clearImageryComponents, 450
  - clearTextComponents, 451
  - getMasterColours, 451
  - getName, 452
  - ImagerySection, 449
  - initMasterColourRect, 453
  - render, 450
  - setMasterColours, 452
  - setMasterColoursPropertyIsColourRect, 452
  - setMasterColoursPropertySource, 452
  - writeXMLToStream, 452
- CEGUI::Imageset, 454
  - defineImage, 461
  - draw, 461, 462
  - getDefaultResourceGroup, 464
  - getImage, 458
  - getImageCount, 458
  - getImageHeight, 459
  - getImageOffset, 460
  - getImageOffsetX, 460
  - getImageOffsetY, 460
  - getImageSize, 459
  - getImageWidth, 459
  - getName, 458
  - getNativeResolution, 463
  - getTexture, 458
  - isAutoScaled, 462
  - isImageDefined, 458
  - load, 464
  - notifyScreenResolution, 463
  - setAutoScalingEnabled, 463
  - setDefaultResourceGroup, 464
  - setNativeResolution, 463
  - setTexture, 464
  - undefineAllImages, 459
  - undefineImage, 458
  - updateImageScalingFactors, 465
  - writeXMLToStream, 463
- CEGUI::Imageset\_xmlHandler, 466
  - Imageset\_xmlHandler, 467
- CEGUI::ImagesetManager, 468
  - createImageset, 469
  - createImagesetFromImageFile, 469
  - destroyAllImagesets, 470
  - destroyImageset, 470
  - getImageset, 470
  - isImagesetPresent, 471
  - notifyScreenResolution, 471
  - writeImagesetToStream, 471
- CEGUI::InvalidRequestException, 478
  - InvalidRequestException, 478
- CEGUI::ItemEntry, 480
  - d\_ownerList, 485
  - getItemPixelSize, 483



- onMouseClicked, 484
- setSelectable, 483
- setSelected, 483
- testClassName\_impl, 484
- validateWindowRenderer, 484
- CEGUI::ItemEntryProperties, 60
- CEGUI::ItemEntryProperties::Selectable, 879
  - get, 880
  - set, 880
- CEGUI::ItemEntryProperties::Selected, 883
  - get, 884
  - set, 884
- CEGUI::ItemEntryWindowRenderer, 486
  - getItemPixelSize, 488
- CEGUI::ItemListBase, 489
  - addItem, 496
  - d\_listItems, 500
  - findItemWithText, 495
  - getContentPane, 498
  - getContentSize, 499
  - getItemCount, 494
  - getItemFromIndex, 494
  - getItemIndex, 494
  - getItemRenderArea, 498
  - handleUpdatedItemData, 497
  - initialiseComponents, 495
  - insertItem, 496
  - isAutoResizeEnabled, 495
  - isItemInList, 495
  - layoutItemWidgets, 499
  - performChildWindowLayout, 497
  - removeItem, 496
  - resetList, 496
  - resetList\_impl, 499
  - setAutoResizeEnabled, 497
  - setSortCallback, 498
  - setSortMode, 498
  - sizeToContent, 497
  - sizeToContent\_impl, 499
  - sortList, 498
  - testClassName\_impl, 500
  - validateWindowRenderer, 500
- CEGUI::ItemListBaseProperties, 61
- CEGUI::ItemListBaseProperties::AutoResizeEnabled, 100
  - get, 101
  - set, 101
- CEGUI::ItemListBaseProperties::SortEnabled, 938
  - get, 939
  - set, 939
- CEGUI::ItemListBaseProperties::SortMode, 942
  - get, 943
  - set, 943
- CEGUI::ItemListBaseWindowRenderer, 501
  - getItemRenderArea, 503
- CEGUI::ItemListbox, 504
  - findSelectedItem, 508
  - getFirstSelectedItem, 506
  - getLastSelectedItem, 506
  - getNextSelectedItem, 507
  - getNextSelectedItemAfter, 507
  - onKeyDown, 508
  - selectRange, 507
  - testClassName\_impl, 507
- CEGUI::ItemListboxProperties, 62
- CEGUI::ItemListboxProperties::MultiSelect, 699
  - get, 700
  - set, 700
- CEGUI::Key, 515
- CEGUI::KeyEventArgs, 517
- CEGUI::LayerSpecification, 518
  - addSectionSpecification, 519
  - clearSectionSpecifications, 519
  - getLayerPriority, 519
  - LayerSpecification, 518
  - render, 519
  - writeXMLToStream, 520
- CEGUI::Listbox, 521
  - addItem, 529
  - clearAllSelections, 530
  - clearAllSelections\_impl, 534
  - configureScrollbars, 533
  - ensureItemIsVisible, 532
  - findItemWithText, 528
  - getFirstSelectedItem, 526
  - getHorzScrollbar, 533
  - getItemAtPoint, 534
  - getItemCount, 526
  - getItemIndex, 527
  - getListboxItemFromIndex, 527
  - getListRenderArea, 533
  - getNextSelected, 526
  - getSelectedCount, 526
  - getVertScrollbar, 533
  - handleUpdatedItemData, 532
  - initialiseComponents, 529
  - insertItem, 529
  - isHorzScrollbarAlwaysShown, 529
  - isItemSelected, 528
  - isListboxItemInList, 528
  - isMultiselectEnabled, 527
  - isSortEnabled, 527
  - isVertScrollbarAlwaysShown, 528
  - onMouseButtonDown, 535
  - onMouseMove, 535
  - onMouseWheel, 535
  - onSized, 535
  - removeItem, 530

- resetList, 529
- resetList\_impl, 534
- setItemSelectState, 531, 532
- setMultiselectEnabled, 530
- setShowHorzScrollbar, 531
- setShowVertScrollbar, 531
- setSortingEnabled, 530
- testClassName\_impl, 534
- validateWindowRenderer, 535
- CEGUI::ListboxItem, 537
  - draw, 545
  - getID, 540
  - getOwnerWindow, 541
  - getPixelSize, 545
  - getSelectionBrushImage, 542
  - getSelectionColours, 541
  - getText, 540
  - getUserData, 540
  - isAutoDeleted, 541
  - isDisabled, 541
  - isSelected, 541
  - setAutoDeleted, 543
  - setDisabled, 543
  - setID, 542
  - setOwnerWindow, 543
  - setSelected, 543
  - setSelectionBrushImage, 544, 545
  - setSelectionColours, 544
  - setText, 542
  - setUserData, 542
- CEGUI::ListboxProperties, 63
- CEGUI::ListboxProperties::ForceHorzScrollbar, 344
  - get, 345
  - set, 345
- CEGUI::ListboxProperties::ForceVertScrollbar, 356
  - get, 357
  - set, 357
- CEGUI::ListboxProperties::ItemTooltips, 513
  - get, 514
  - set, 514
- CEGUI::ListboxProperties::MultiSelect, 701
  - get, 702
  - set, 702
- CEGUI::ListboxProperties::Sort, 926
  - get, 927
  - set, 927
- CEGUI::ListboxTextItem, 546
  - draw, 549
  - getFont, 547
  - getPixelSize, 549
  - getTextColours, 547
  - setFont, 548
  - setTextColours, 548, 549
- CEGUI::ListboxWindowRenderer, 550
  - getListRenderArea, 552
- CEGUI::ListHeader, 553
  - addColumn, 565
  - createInitialisedSegment, 568
  - createNewSegment, 569
  - destroyListSegment, 569
  - getColumnCount, 559
  - getColumnFromID, 560
  - getColumnFromSegment, 560
  - getColumnWidth, 562
  - getColumnWithText, 561
  - getPixelOffsetToColumn, 561
  - getPixelOffsetToSegment, 561
  - getSegmentFromColumn, 559
  - getSegmentFromID, 559
  - getSegmentOffset, 563
  - getSortColumn, 560
  - getSortDirection, 562
  - getSortSegment, 559
  - getTotalSegmentsPixelExtent, 561
  - insertColumn, 565
  - isColumnDraggingEnabled, 562
  - isColumnSizingEnabled, 562
  - isSortingEnabled, 562
  - moveColumn, 566, 567
  - moveSegment, 567
  - removeColumn, 566
  - removeSegment, 566
  - setColumnDraggingEnabled, 564
  - setColumnSizingEnabled, 564
  - setColumnWidth, 568
  - setSegmentOffset, 568
  - setSortColumn, 564
  - setSortColumnFromID, 564
  - setSortDirection, 563
  - setSortingEnabled, 563
  - setSortSegment, 563
  - testClassName\_impl, 569
  - validateWindowRenderer, 570
- CEGUI::ListHeaderProperties, 64
- CEGUI::ListHeaderProperties::ColumnsMovable, 144
  - get, 145
  - set, 145
- CEGUI::ListHeaderProperties::ColumnsSizable, 148
  - get, 149
  - set, 149
- CEGUI::ListHeaderProperties::SortColumnID, 930
  - get, 931
  - set, 931
- CEGUI::ListHeaderProperties::SortDirection, 934

- get, 935
  - set, 935
- CEGUI::ListHeaderProperties::SortSettingEnabled, 946
  - get, 947
  - set, 947
- CEGUI::ListHeaderSegment, 571
  - Ascending, 576
  - Descending, 576
  - doDragMoving, 578
  - doDragSizing, 578
  - getDragMoveOffset, 576
  - getSortDirection, 576
  - isClickable, 577
  - isDragMoveThresholdExceeded, 578
  - isDragMovingEnabled, 576
  - isSizingEnabled, 576
  - None, 576
  - onCaptureLost, 580
  - onMouseButtonDown, 579
  - onMouseButtonUp, 579
  - onMouseDoubleClicked, 580
  - onMouseLeaves, 580
  - onMouseMove, 579
  - setClickable, 578
  - setDragMovingEnabled, 577
  - setSizingEnabled, 577
  - setSortDirection, 577
  - SortDirection, 576
  - testClassName\_impl, 579
- CEGUI::ListHeaderSegmentProperties, 65
- CEGUI::ListHeaderSegmentProperties::Clickable, 124
  - get, 125
  - set, 125
- CEGUI::ListHeaderSegmentProperties::Dragable, 222
  - get, 223
  - set, 223
- CEGUI::ListHeaderSegmentProperties::MovingCursorImage, 638
  - get, 638
  - set, 639
- CEGUI::ListHeaderSegmentProperties::Sizable, 904
  - get, 905
  - set, 905
- CEGUI::ListHeaderSegmentProperties::SizingCursorImage, 909
  - get, 909
  - set, 910
- CEGUI::ListHeaderSegmentProperties::SortDirection, 932
  - get, 933
  - set, 933
- CEGUI::ListHeaderWindowRenderer, 581
  - createNewSegment, 583
  - destroyListSegment, 583
- CEGUI::Logger, 584
  - getLoggingLevel, 585
  - logEvent, 585
  - setLogFilename, 585
  - setLoggingLevel, 585
- CEGUI::MCLGridRef, 603
- CEGUI::MemberFunctionSlot, 604
- CEGUI::MemoryException, 605
  - MemoryException, 605
- CEGUI::Menubar, 607
  - getContentSize, 608
  - layoutItemWidgets, 608
  - testClassName\_impl, 608
- CEGUI::MenuBase, 610
  - changePopupMenuItem, 612
  - getItemSpacing, 611
  - getPopupMenuItem, 612
  - isMultiplePopupsAllowed, 611
  - testClassName\_impl, 612
- CEGUI::MenuBaseProperties::AllowMultiplePopups, 88
  - get, 89
  - set, 89
- CEGUI::MenuBaseProperties::ItemSpacing, 509
  - get, 510
  - set, 510
- CEGUI::MenuItem, 613
  - closeAllMenuItemPopups, 618
  - closePopupMenu, 616
  - getPopupMenu, 616
  - isHovering, 615
  - isPushed, 615
  - onCaptureLost, 617
  - onMouseButtonDown, 617
  - onMouseButtonUp, 617
  - onMouseLeaves, 617
  - onMouseMove, 617
  - onTextChanged, 618
  - openPopupMenu, 616
  - setPopupMenu, 616
  - setPopupMenu\_impl, 618
  - testClassName\_impl, 619
  - togglePopupMenu, 616
  - updateInternalState, 618
- CEGUI::MouseClickedTracker, 624
- CEGUI::MouseCursor, 625
  - draw, 628
  - getConstraintArea, 630
  - getDisplayIndependantPosition, 630
  - getImage, 627

- getPosition, 629
- getSingleton, 627
- getSingletonPtr, 627
- getUnifiedConstraintArea, 630
- hide, 629
- isVisible, 629
- offsetPosition, 628
- setConstraintArea, 628
- setImage, 627
- setPosition, 628
- setUnifiedConstraintArea, 628
- setVisible, 629
- show, 629
- CEGUI::MouseCursorEventArgs, 631
- CEGUI::MouseEventArgs, 634
- CEGUI::MultiColumnList, 640
  - addColumn, 660
  - addRow, 662
  - autoSizeColumnHeader, 670
  - clearAllSelections, 667
  - clearAllSelections\_impl, 670
  - configureScrollbars, 670
  - findColumnItemWithText, 654
  - findListItemWithText, 655
  - findRowItemWithText, 655
  - getColumnCount, 650
  - getColumnHeaderWidth, 651
  - getColumnID, 658
  - getColumnWithHeaderText, 651
  - getColumnWithID, 651
  - getFirstSelectedItem, 656
  - getHeaderSegmentForColumn, 652
  - getHorzScrollbar, 659
  - getItemAtGridReference, 653
  - getItemAtPoint, 671
  - getItemColumnIndex, 652
  - getItemGridReference, 653
  - getItemRowIndex, 652
  - getListHeader, 659
  - getListRenderArea, 659
  - getNextSelected, 656
  - getNominatedSelectionColumn, 657
  - getNominatedSelectionColumnID, 657
  - getNominatedSelectionRow, 657
  - getRowCount, 650
  - getRowID, 658
  - getRowWithID, 659
  - getSelectedCount, 656
  - getSelectionMode, 657
  - getSortColumn, 650
  - getSortDirection, 652
  - getTotalColumnHeadersWidth, 651
  - getVertScrollbar, 659
  - handleUpdatedItemData, 668
  - initialiseComponents, 660
  - insertColumn, 660
  - insertRow, 663
  - isHorzScrollbarAlwaysShown, 658
  - isItemSelected, 656
  - isListBoxItemInColumn, 653
  - isListBoxItemInList, 654
  - isListBoxItemInRow, 654
  - isUserColumnDraggingEnabled, 650
  - isUserColumnSizingEnabled, 650
  - isUserSortControlEnabled, 650
  - isVertScrollbarAlwaysShown, 658
  - moveColumn, 661
  - moveColumn\_impl, 671
  - moveColumnWithID, 662
  - onFontChanged, 672
  - onMouseDown, 672
  - onMouseWheel, 672
  - onSized, 672
  - removeColumn, 661
  - removeColumnWithID, 661
  - removeRow, 664
  - resetList, 660
  - resetList\_impl, 671
  - setColumnHeaderWidth, 669
  - setItem, 664
  - setItemSelectState, 668
  - setNominatedSelectionColumn, 665
  - setNominatedSelectionColumnID, 665
  - setNominatedSelectionRow, 666
  - setRowID, 670
  - setSelectionMode, 665
  - setShowHorzScrollbar, 667
  - setShowVertScrollbar, 667
  - setSortColumn, 666
  - setSortColumnByID, 667
  - setSortDirection, 666
  - setUserColumnDraggingEnabled, 669
  - setUserColumnSizingEnabled, 669
  - setUserSortControlEnabled, 669
  - testClassName\_impl, 671
  - validateWindowRenderer, 671
- CEGUI::MultiColumnList::ListRow, 674
- CEGUI::MultiColumnListProperties, 66
  - CEGUI::MultiColumnListProperties::ColumnHeader, 142
    - get, 143
    - set, 143
  - CEGUI::MultiColumnListProperties::ColumnsMovable, 146
    - get, 147
    - set, 147
  - CEGUI::MultiColumnListProperties::ColumnsSizable, 150

- get, [151](#)
- set, [151](#)
- CEGUI::MultiColumnListProperties::ForceHorzScrollbar, [346](#)
- get, [347](#)
- set, [347](#)
- CEGUI::MultiColumnListProperties::ForceVertScrollbar, [360](#)
- get, [361](#)
- set, [361](#)
- CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, [707](#)
- get, [708](#)
- set, [708](#)
- CEGUI::MultiColumnListProperties::NominatedSelectionRow, [709](#)
- get, [710](#)
- set, [710](#)
- CEGUI::MultiColumnListProperties::RowCount, [812](#)
- get, [812](#)
- set, [813](#)
- CEGUI::MultiColumnListProperties::SelectionMode, [895](#)
- get, [896](#)
- set, [896](#)
- CEGUI::MultiColumnListProperties::SortColumnID, [928](#)
- get, [929](#)
- set, [929](#)
- CEGUI::MultiColumnListProperties::SortDirection, [936](#)
- get, [937](#)
- set, [937](#)
- CEGUI::MultiColumnListProperties::SortSettingEnabled, [944](#)
- get, [945](#)
- set, [945](#)
- CEGUI::MultiColumnListWindowRenderer, [675](#)
- getListRenderArea, [677](#)
- CEGUI::MultiLineEditbox, [678](#)
- eraseSelectedText, [689](#)
- formatText, [688](#)
- getCaratIndex, [685](#)
- getHorzScrollbar, [686](#)
- getMaxTextLength, [685](#)
- getNextTokenLength, [689](#)
- getSelectionEndIndex, [685](#)
- getSelectionLength, [685](#)
- getSelectionStartIndex, [685](#)
- getTextIndexFromPosition, [689](#)
- getTextRenderArea, [686](#)
- getVertScrollbar, [686](#)
- hasInputFocus, [684](#)
- initialiseComponents, [687](#)
- isReadOnly, [685](#)
- isVertScrollbarAlwaysShown, [686](#)
- isWordWrapped, [686](#)
- onCaptureLost, [691](#)
- onCharacter, [691](#)
- onKeyDown, [692](#)
- onMouseButtonDown, [690](#)
- onMouseButtonUp, [690](#)
- onMouseDoubleClicked, [690](#)
- onMouseMove, [691](#)
- onMouseTripleClicked, [691](#)
- onMouseWheel, [692](#)
- onSized, [692](#)
- onTextChanged, [692](#)
- setCaratIndex, [687](#)
- setMaxTextLength, [688](#)
- setReadOnly, [687](#)
- setSelection, [687](#)
- setShowVertScrollbar, [688](#)
- setWordWrapping, [688](#)
- testClassName\_impl, [689](#)
- validateWindowRenderer, [690](#)
- CEGUI::MultiLineEditbox::LineInfo, [693](#)
- CEGUI::MultiLineEditboxProperties, [67](#)
- CEGUI::MultiLineEditboxProperties::CaratIndex, [113](#)
- get, [114](#)
- set, [114](#)
- CEGUI::MultiLineEditboxProperties::ForceVertScrollbar, [348](#)
- get, [349](#)
- set, [349](#)
- CEGUI::MultiLineEditboxProperties::MaxTextLength, [599](#)
- get, [600](#)
- set, [600](#)
- CEGUI::MultiLineEditboxProperties::ReadOnly, [780](#)
- get, [781](#)
- set, [781](#)
- CEGUI::MultiLineEditboxProperties::SelectionBrushImage, [889](#)
- get, [889](#)
- set, [890](#)
- CEGUI::MultiLineEditboxProperties::SelectionLength, [891](#)
- get, [892](#)
- set, [892](#)
- CEGUI::MultiLineEditboxProperties::SelectionStart, [899](#)
- get, [900](#)
- set, [900](#)

- CEGUI::MultiLineEditboxProperties::WordWrap, 1326
  - get, 1327
  - set, 1327
- CEGUI::MultiLineEditboxWindowRenderer, 694
  - getTextRenderArea, 696
- CEGUI::NamedArea, 703
  - getArea, 703
  - getName, 703
  - setArea, 704
  - writeXMLToStream, 704
- CEGUI::NullObjectException, 715
  - NullObjectException, 715
- CEGUI::ObjectInUseException, 719
  - ObjectInUseException, 719
- CEGUI::PixmapFont, 725
  - PixmapFont, 727
- CEGUI::PopupMenu, 728
  - closePopupMenu, 731
  - getContentSize, 732
  - getFadeInTime, 730
  - getFadeOutTime, 730
  - layoutItemWidgets, 731
  - onAlphaChanged, 732
  - onDestructionStarted, 732
  - onHidden, 733
  - onMouseButtonDown, 733
  - onMouseButtonUp, 733
  - onShown, 733
  - openPopupMenu, 731
  - setFadeInTime, 730
  - setFadeOutTime, 731
  - testClassName\_impl, 732
  - updateSelf, 731
- CEGUI::PopupMenuProperties::FadeInTime, 292
  - get, 293
  - set, 293
- CEGUI::PopupMenuProperties::FadeOutTime, 294
  - get, 295
  - set, 295
- CEGUI::ProgressBar, 734
  - adjustProgress, 737
  - setProgress, 737
  - setStepSize, 737
  - step, 737
  - testClassName\_impl, 738
- CEGUI::ProgressBarProperties, 68
- CEGUI::ProgressBarProperties::CurrentProgress, 199
  - get, 200
  - set, 200
- CEGUI::ProgressBarProperties::StepSize, 962
  - get, 963
  - set, 963
- CEGUI::Property, 739
  - get, 742
  - getDefault, 746
  - getHelp, 742
  - getName, 742
  - isDefault, 746
  - Property, 742
  - set, 744
  - writeXMLToStream, 747
- CEGUI::PropertyDefinition, 748
  - get, 749
  - set, 749
  - writeXMLElementType, 749
- CEGUI::PropertyDefinitionBase, 751
  - set, 752
  - writeXMLAttributes, 753
  - writeXMLElementType, 752
  - writeXMLToStream, 752
- CEGUI::PropertyDim, 754
  - PropertyDim, 755
- CEGUI::PropertyHelper, 756
- CEGUI::PropertyInitialiser, 757
  - apply, 758
  - getInitialiserValue, 758
  - getTargetPropertyName, 758
  - PropertyInitialiser, 757
  - writeXMLToStream, 758
- CEGUI::PropertyLinkDefinition, 759
  - get, 760
  - getTargetWindow, 761
  - set, 760
  - writeXMLAttributes, 761
  - writeXMLElementType, 761
- CEGUI::PropertyReceiver, 762
- CEGUI::PropertySet, 763
  - addProperty, 764
  - clearProperties, 764
  - getProperty, 765
  - getPropertyDefault, 766
  - getPropertyHelp, 765
  - isPropertyDefault, 766
  - isPropertyPresent, 765
  - removeProperty, 764
  - setProperty, 765
- CEGUI::PushButton, 767
  - onMouseButtonUp, 768
  - testClassName\_impl, 768
- CEGUI::RadioButton, 770
  - getGroupID, 771
  - getSelectedButtonInGroup, 772
  - isSelected, 771
  - onMouseButtonUp, 773
  - setGroupID, 772
  - setSelected, 772

- testClassName\_impl, 772
- CEGUI::RadioButtonProperties, 69
- CEGUI::RadioButtonProperties::GroupID, 403
  - get, 404
  - set, 404
- CEGUI::RadioButtonProperties::Selected, 881
  - get, 882
  - set, 882
- CEGUI::RawDataContainer, 774
  - getDataPtr, 774
  - getSize, 775
  - setData, 774
  - setSize, 774
- CEGUI::Rect, 782
  - constrainSize, 784
  - constrainSizeMax, 784
  - constrainSizeMin, 784
  - getIntersection, 783
  - isPointInRect, 783
  - offset, 783
- CEGUI::RefCounted, 785
- CEGUI::RegexValidator, 787
- CEGUI::RenderCache, 788
  - cacheImage, 789
  - cacheText, 789
  - hasCachedImagery, 789
  - render, 789
- CEGUI::Renderer, 791
  - addQuad, 793
  - advanceZValue, 797
  - clearRenderList, 794
  - createTexture, 794, 795
  - destroyAllTextures, 795
  - destroyTexture, 795
  - doRender, 793
  - EventDisplaySizeChanged, 798
  - getCurrentZ, 797
  - getHeight, 796
  - getHorzScreenDPI, 796
  - getIdentifierString, 797
  - getMaxTextureSize, 796
  - getRect, 796
  - getSize, 796
  - getVertScreenDPI, 797
  - getWidth, 796
  - getZLayer, 797
  - isQueueingEnabled, 795
  - resetZValue, 797
  - setQueueingEnabled, 794
- CEGUI::RendererException, 799
  - RendererException, 799
- CEGUI::ResourceProvider, 801
  - getDefaultResourceGroup, 802
  - loadRawDataContainer, 802
  - setDefaultResourceGroup, 802
  - unloadRawDataContainer, 802
- CEGUI::Scheme, 814
  - ~Scheme, 815
  - getDefaultResourceGroup, 816
  - getName, 816
  - loadResources, 815
  - resourcesLoaded, 815
  - setDefaultResourceGroup, 816
  - unloadResources, 815
- CEGUI::Scheme\_xmlHandler, 817
  - Scheme\_xmlHandler, 818
- CEGUI::SchemeManager, 819
  - getScheme, 820
  - isSchemePresent, 820
  - loadScheme, 819
  - unloadAllSchemes, 820
  - unloadScheme, 820
- CEGUI::ScriptException, 822
  - ScriptException, 822
- CEGUI::ScriptFunctor, 824
- CEGUI::ScriptModule, 825
  - createBindings, 827
  - destroyBindings, 828
  - executeScriptedEventHandler, 827
  - executeScriptFile, 826
  - executeScriptGlobal, 827
  - executeString, 827
  - getDefaultResourceGroup, 829
  - getIdentifierString, 828
  - setDefaultResourceGroup, 829
  - subscribeEvent, 828
- CEGUI::ScriptWindowHelper, 830
  - getWindow, 831
- CEGUI::ScrollablePane, 832
  - configureScrollbars, 842
  - destroy, 842
  - getContentPane, 837
  - getContentPaneArea, 838
  - getHorizontalOverlapSize, 839
  - getHorizontalScrollPosition, 839
  - getHorizontalStepSize, 839
  - getHorzScrollbar, 841
  - getScrolledContainer, 843
  - getVerticalOverlapSize, 840
  - getVerticalScrollPosition, 841
  - getVerticalStepSize, 840
  - getVertScrollbar, 841
  - getViewableArea, 841
  - initialiseComponents, 842
  - isContentPaneAutoSized, 838
  - isHorzScrollbarAlwaysShown, 837
  - isHorzScrollbarNeeded, 843
  - isVertScrollbarAlwaysShown, 837



- isVertScrollbarNeeded, 842
- onAutoSizeSettingChanged, 844
- onContentPaneChanged, 844
- onContentPaneScrolled, 845
- onHorzScrollbarModeChanged, 844
- onMouseWheel, 845
- onSized, 845
- onVertScrollbarModeChanged, 844
- setContentPaneArea, 838
- setContentPaneAutoSized, 838
- setHorizontalOverlapSize, 839
- setHorizontalScrollPosition, 840
- setHorizontalStepSize, 839
- setShowHorzScrollbar, 837
- setShowVertScrollbar, 837
- setVerticalOverlapSize, 840
- setVerticalScrollPosition, 841
- setVerticalStepSize, 840
- testClassName\_impl, 843
- validateWindowRenderer, 843
- CEGUI::ScrollablePaneProperties, 70
- CEGUI::ScrollablePaneProperties::ContentArea, 185
  - get, 186
  - set, 186
- CEGUI::ScrollablePaneProperties::ContentPaneAutoSized, 189
  - get, 190
  - set, 190
- CEGUI::ScrollablePaneProperties::ForceHorzScrollbar, 336
  - get, 337
  - set, 337
- CEGUI::ScrollablePaneProperties::ForceVertScrollbar, 350
  - get, 351
  - set, 351
- CEGUI::ScrollablePaneProperties::HorzOverlapSize, 416
  - get, 417
  - set, 417
- CEGUI::ScrollablePaneProperties::HorzScrollPosition, 420
  - get, 421
  - set, 421
- CEGUI::ScrollablePaneProperties::HorzStepSize, 422
  - get, 423
  - set, 423
- CEGUI::ScrollablePaneProperties::VertOverlapSize, 1192
  - get, 1193
  - set, 1193
- CEGUI::ScrollablePaneProperties::VertScrollPosition, 1196
  - get, 1197
  - set, 1197
- CEGUI::ScrollablePaneProperties::VertStepSize, 1198
  - get, 1199
  - set, 1199
- CEGUI::ScrollablePaneWindowRenderer, 846
  - getViewableArea, 848
- CEGUI::Scrollbar, 849
  - getAdjustDirectionFromPoint, 857
  - getDecreaseButton, 854
  - getDocumentSize, 852
  - getIncreaseButton, 854
  - getOverlapSize, 853
  - getPageSize, 853
  - getScrollPosition, 854
  - getStepSize, 853
  - getThumb, 854
  - getValueFromThumb, 857
  - handleThumbMoved, 857
  - initialiseComponents, 854
  - onMouseButtonDown, 858
  - onMouseWheel, 858
  - setDocumentSize, 855
  - setOverlapSize, 856
  - setPageSize, 855
  - setScrollPosition, 856
  - setStepSize, 855
  - testClassName\_impl, 857
  - validateWindowRenderer, 858
- CEGUI::ScrollbarProperties, 71
- CEGUI::ScrollbarProperties::DocumentSize, 220
  - get, 221
  - set, 221
- CEGUI::ScrollbarProperties::OverlapSize, 721
  - get, 722
  - set, 722
- CEGUI::ScrollbarProperties::PageSize, 723
  - get, 724
  - set, 724
- CEGUI::ScrollbarProperties::ScrollPosition, 871
  - get, 872
  - set, 872
- CEGUI::ScrollbarProperties::StepSize, 966
  - get, 967
  - set, 967
- CEGUI::ScrollbarWindowRenderer, 859
  - getAdjustDirectionFromPoint, 861
  - getValueFromThumb, 861
- CEGUI::ScrolledContainer, 862
  - drawSelf, 866
  - getChildExtentsArea, 865



- getContentArea, 865
- getUnclippedInnerRect\_impl, 865
- isContentPaneAutoSized, 864
- onAutoSizeSettingChanged, 866
- onChildAdded, 866
- onChildRemoved, 867
- onContentChanged, 866
- onParentSized, 867
- setContentArea, 865
- setContentPaneAutoSized, 864
- testClassName\_impl, 865
- CEGUI::ScrolledContainerProperties, 72
- CEGUI::ScrolledContainerProperties::ChildExtentsArea, 122
  - get, 123
  - set, 123
- CEGUI::ScrolledContainerProperties::ContentArea, 187
  - get, 188
  - set, 188
- CEGUI::ScrolledContainerProperties::ContentPaneAutoSized, 191
  - get, 192
  - set, 192
- CEGUI::ScrolledItemListBase, 868
  - initialiseComponents, 870
  - onMouseWheel, 870
  - testClassName\_impl, 870
- CEGUI::ScrolledItemListBaseProperties, 73
- CEGUI::ScrolledItemListBaseProperties::ForceHorizontalScrollbar, 338
  - get, 339
  - set, 339
- CEGUI::ScrolledItemListBaseProperties::ForceVerticalScrollbar, 352
  - get, 353
  - set, 353
- CEGUI::SectionSpecification, 873
  - getOverrideColours, 876
  - getOwnerWidgetLookFeel, 875
  - getSectionName, 875
  - initColourRectForOverride, 877
  - isUsingOverrideColours, 876
  - render, 875
  - SectionSpecification, 874
  - setOverrideColours, 876
  - setOverrideColoursPropertyIsColourRect, 877
  - setOverrideColoursPropertySource, 876
  - setRenderControlPropertySource, 877
  - setUsingOverrideColours, 876
  - writeXMLToStream, 877
- CEGUI::SimpleTimer, 901
- CEGUI::Size, 906
- CEGUI::Slider, 913
  - getAdjustDirectionFromPoint, 917
  - getClickStep, 916
  - getCurrentValue, 916
  - getMaxValue, 916
  - getThumb, 916
  - getValueFromThumb, 917
  - handleThumbMoved, 918
  - initialiseComponents, 916
  - onMouseButtonDown, 919
  - onMouseWheel, 919
  - setClickStep, 917
  - setCurrentValue, 917
  - setMaxValue, 916
  - testClassName\_impl, 918
  - validateWindowRenderer, 918
- CEGUI::SliderProperties, 74
- CEGUI::SliderProperties::ClickStepSize, 126
  - get, 127
  - set, 127
- CEGUI::SliderProperties::CurrentValue, 201
  - get, 202
  - set, 202
- CEGUI::SliderProperties::MaximumValue, 595
  - get, 596
  - set, 596
- CEGUI::SliderWindowRenderer, 920
  - getAdjustDirectionFromPoint, 922
  - getValueFromThumb, 922
- CEGUI::SlotFunctorBase, 923
- CEGUI::Spinner, 948
  - FloatingPoint, 952
  - getCurrentValue, 952
  - getDecreaseButton, 955
  - getEditbox, 956
  - getIncreaseButton, 955
  - getMaximumValue, 953
  - getMinimumValue, 953
  - getStepSize, 952
  - getTextFromValue, 955
  - getTextInputMode, 953
  - getValueFromText, 954
  - Hexadecimal, 952
  - initialiseComponents, 952
  - Integer, 952
  - Octal, 952
  - onActivated, 956
  - onFontChanged, 956
  - onMaximumValueChanged, 957
  - onMinimumValueChanged, 957
  - onStepChanged, 957
  - onTextChanged, 956
  - onTextInputModeChanged, 958
  - onValueChanged, 957
  - setCurrentValue, 953

- setMaximumValue, 954
  - setMinimumValue, 954
  - setStepSize, 953
  - setTextInputMode, 954
  - testClassName\_impl, 955
  - TextInputMode, 952
- CEGUI::SpinnerProperties::CurrentValue, 203
  - get, 204
  - set, 204
- CEGUI::SpinnerProperties::MaximumValue, 597
  - get, 598
  - set, 598
- CEGUI::SpinnerProperties::MinimumValue, 620
  - get, 621
  - set, 621
- CEGUI::SpinnerProperties::StepSize, 964
  - get, 965
  - set, 965
- CEGUI::SpinnerProperties::TextInputMode, 1089
  - get, 1090
  - set, 1090
- CEGUI::StateImagery, 959
  - addLayer, 960
  - clearLayers, 960
  - getName, 961
  - isClippedToDisplay, 961
  - render, 960
  - setClippedToDisplay, 961
  - StateImagery, 960
  - writeXMLToStream, 961
- CEGUI::String, 968
  - append, 995–1000
  - assign, 991–995
  - at, 989
  - begin, 1033
  - c\_str, 989
  - capacity, 982
  - clear, 1005
  - compare, 983–988
  - copy, 990
  - data, 990
  - empty, 982
  - end, 1033
  - erase, 1005, 1006
  - find, 1015–1020
  - find\_first\_not\_of, 1021–1026
  - find\_first\_of, 1021–1026
  - find\_last\_not\_of, 1027–1032
  - find\_last\_of, 1027–1032
  - insert, 1000–1005
  - length, 982
  - max\_size, 982
  - operator+=, 995–999
  - operator=, 991–994
  - push\_back, 998
  - rbegin, 1033, 1034
  - rend, 1034
  - replace, 1007–1015
  - reserve, 983
  - resize, 1006, 1007
  - rfind, 1015–1020
  - size, 982
  - String, 978–981
  - substr, 1032
  - swap, 995
  - utf8\_stream\_len, 990
- CEGUI::String::const\_iterator, 1035
- CEGUI::String::FastLessCompare, 1036
- CEGUI::String::iterator, 1037
- CEGUI::SubComp, 1038
- CEGUI::SubscriberSlot, 1039
- CEGUI::System, 1041
  - executeScriptFile, 1051
  - executeScriptGlobal, 1051
  - executeScriptString, 1051
  - getDefaultFont, 1046
  - getDefaultMouseCursor, 1049
  - getDefaultTooltip, 1053
  - getDefaultXMLParserName, 1054
  - getGUISheet, 1047
  - getModalTarget, 1053
  - getMouseMoveScaling, 1052
  - getMultiClickTimeout, 1048
  - getMultiClickToleranceAreaSize, 1048
  - getRenderer, 1046
  - getResourceProvider, 1051
  - getScriptingModule, 1050
  - getSingleClickTimeout, 1048
  - getSingleton, 1046
  - getSingletonPtr, 1046
  - getSystemKeys, 1052
  - getWindowContainingMouse, 1050
  - injectChar, 1056
  - injectKeyDown, 1056
  - injectKeyUp, 1056
  - injectMouseButtonDown, 1055
  - injectMouseButtonUp, 1055
  - injectMouseLeaves, 1055
  - injectMouseMove, 1055
  - injectMousePosition, 1057
  - injectMouseWheelChange, 1056
  - injectTimePulse, 1057
  - isRedrawRequested, 1047
  - notifyWindowDestroyed, 1052
  - renderGUI, 1047
  - setDefaultFont, 1046
  - setDefaultMouseCursor, 1049, 1050
  - setDefaultTooltip, 1052, 1053

- setDefaultXMLParserName, 1054
- setGUISheet, 1047
- setModalTarget, 1053
- setMouseMoveScaling, 1052
- setMultiClickTimeout, 1048
- setMultiClickToleranceAreaSize, 1049
- setScriptingModule, 1050
- setSingleClickTimeout, 1048
- signalRedraw, 1047
- System, 1045
- CEGUI::TabButton, 1058
  - onMouseDown, 1060
  - onMouseUp, 1060
  - onMouseMove, 1061
  - onMouseWheel, 1060
  - testClassName\_impl, 1061
- CEGUI::TabControl, 1062
  - addTab, 1069
  - calculateTabButtonSizePosition, 1072
  - createTabButton, 1071
  - drawSelf, 1069
  - getSelectedTabIndex, 1068
  - getTabButtonPane, 1070
  - getTabContents, 1067, 1068
  - getTabContentsAtIndex, 1067
  - getTabCount, 1067
  - getTabPane, 1070
  - getTabPanePosition, 1067
  - initialiseComponents, 1068
  - isTabContentsSelected, 1068
  - makeTabVisible\_impl, 1070
  - onFontChanged, 1071
  - performChildWindowLayout, 1071
  - removeTab, 1069
  - selectTab\_impl, 1069
  - setTabPanePosition, 1067
  - testClassName\_impl, 1070
  - validateWindowRenderer, 1071
- CEGUI::TabControlProperties, 75
- CEGUI::TabControlProperties::TabHeight, 1076
  - get, 1076
  - set, 1077
- CEGUI::TabControlProperties::TabPanePosition, 1078
  - get, 1078
  - set, 1079
- CEGUI::TabControlProperties::TabTextPadding, 1080
  - get, 1080
  - set, 1081
- CEGUI::TabControlWindowRenderer, 1073
  - createTabButton, 1075
- CEGUI::TextComponent, 1084
  - getFont, 1086
  - getFontPropertySource, 1088
  - getHorizontalFormatting, 1087
  - getText, 1085
  - getTextPropertySource, 1087
  - getVerticalFormatting, 1086
  - isFontFetchedFromProperty, 1088
  - isTextFetchedFromProperty, 1087
  - setFont, 1086
  - setFontPropertySource, 1088
  - setHorizontalFormatting, 1087
  - setText, 1085
  - setTextPropertySource, 1088
  - setVerticalFormatting, 1086
  - writeXMLToStream, 1087
- CEGUI::Texture, 1091
  - getHeight, 1093
  - getOriginalHeight, 1093
  - getOriginalWidth, 1092
  - getRenderer, 1094
  - getWidth, 1092
  - getXScale, 1093
  - getYScale, 1093
  - loadFromFile, 1093
  - loadFromMemory, 1094
  - PF\_RGB, 1092
  - PF\_RGBA, 1092
  - PixelFormat, 1092
- CEGUI::TextUtils, 1095
  - getNextWord, 1096
  - getNextWordStartIdx, 1096
  - getWordStartIdx, 1096
  - trimLeadingChars, 1096
  - trimTrailingChars, 1097
- CEGUI::Thumb, 1098
  - getHorzRange, 1101
  - getVertRange, 1101
  - isHorzFree, 1101
  - isHotTracked, 1100
  - isVertFree, 1101
  - onCaptureLost, 1103
  - onMouseDown, 1103
  - onMouseMove, 1103
  - setHorzFree, 1102
  - setHorzRange, 1102
  - setHotTracked, 1101
  - setVertFree, 1101
  - setVertRange, 1102
  - testClassName\_impl, 1102
- CEGUI::ThumbProperties, 76
- CEGUI::ThumbProperties::HorzFree, 414
  - get, 415
  - set, 415
- CEGUI::ThumbProperties::HorzRange, 418
  - get, 419

- set, [419](#)
- CEGUI::ThumbProperties::HotTracked, [424](#)
  - get, [425](#)
  - set, [425](#)
- CEGUI::ThumbProperties::VertFree, [1188](#)
  - get, [1189](#)
  - set, [1189](#)
- CEGUI::ThumbProperties::VertRange, [1194](#)
  - get, [1195](#)
  - set, [1195](#)
- CEGUI::Titlebar, [1104](#)
  - isDraggingEnabled, [1107](#)
  - onCaptureLost, [1108](#)
  - onDraggingModeChanged, [1108](#)
  - onFontChanged, [1108](#)
  - onMouseButtonDown, [1107](#)
  - onMouseButtonUp, [1108](#)
  - onMouseDoubleClicked, [1108](#)
  - onMouseMove, [1107](#)
  - setDraggingEnabled, [1107](#)
  - testClassName\_impl, [1109](#)
- CEGUI::TitlebarProperties, [77](#)
- CEGUI::TitlebarProperties::DraggingEnabled, [244](#)
  - get, [245](#)
  - set, [245](#)
- CEGUI::Tooltip, [1114](#)
  - Active, [1117](#)
  - FadeIn, [1117](#)
  - FadeOut, [1117](#)
  - getDisplayTime, [1119](#)
  - getFadeTime, [1118](#)
  - getHoverTime, [1118](#)
  - getTargetWindow, [1118](#)
  - getTextSize, [1120](#)
  - getTextSize\_impl, [1120](#)
  - Inactive, [1117](#)
  - onDisplayTimeChanged, [1121](#)
  - onFadeTimeChanged, [1121](#)
  - onHoverTimeChanged, [1120](#)
  - onMouseEnters, [1122](#)
  - onTextChanged, [1122](#)
  - onTooltipActive, [1121](#)
  - onTooltipInactive, [1121](#)
  - positionSelf, [1119](#)
  - resetTimer, [1118](#)
  - setDisplayTime, [1118](#)
  - setFadeTime, [1119](#)
  - setHoverTime, [1119](#)
  - setTargetWindow, [1118](#)
  - sizeSelf, [1119](#)
  - testClassName\_impl, [1120](#)
  - TipState, [1117](#)
  - updateSelf, [1122](#)
  - validateWindowRenderer, [1120](#)
- CEGUI::TooltipProperties, [78](#)
- CEGUI::TooltipProperties::DisplayTime, [216](#)
  - get, [217](#)
  - set, [217](#)
- CEGUI::TooltipProperties::FadeTime, [296](#)
  - get, [297](#)
  - set, [297](#)
- CEGUI::TooltipProperties::HoverTime, [426](#)
  - get, [427](#)
  - set, [427](#)
- CEGUI::TooltipWindowRenderer, [1123](#)
  - getTextSize, [1125](#)
- CEGUI::Tree, [1126](#)
  - addItem, [1135](#)
  - cacheTreeBaseImagery, [1139](#)
  - clearAllSelections, [1136](#)
  - clearAllSelections\_impl, [1140](#)
  - createHorzScrollbar, [1139](#)
  - createVertScrollbar, [1139](#)
  - ensureItemIsVisible, [1138](#)
  - findFirstItemWithID, [1133](#)
  - findFirstItemWithText, [1133](#)
  - getFirstSelectedItem, [1132](#)
  - getHeightToItemInList, [1140](#)
  - getItemAtPoint, [1140](#)
  - getItemCount, [1132](#)
  - getLastSelectedItem, [1132](#)
  - getNextSelected, [1132](#)
  - getSelectedCount, [1132](#)
  - getTreeRenderArea, [1139](#)
  - handleUpdatedItemData, [1138](#)
  - initialise, [1134](#)
  - insertItem, [1135](#)
  - isHorzScrollbarAlwaysShown, [1134](#)
  - isMultiselectEnabled, [1133](#)
  - isSortEnabled, [1133](#)
  - isTreeItemInList, [1134](#)
  - isVertScrollbarAlwaysShown, [1134](#)
  - onMouseButtonDown, [1141](#)
  - onMouseMove, [1141](#)
  - onMouseWheel, [1141](#)
  - onSized, [1141](#)
  - populateRenderCache, [1141](#)
  - removeItem, [1135](#)
  - resetList, [1135](#)
  - resetList\_impl, [1140](#)
  - setItemSelectState, [1137](#)
  - setLookNFeel, [1138](#)
  - setMultiselectEnabled, [1136](#)
  - setShowHorzScrollbar, [1137](#)
  - setShowVertScrollbar, [1136](#)
  - setSortingEnabled, [1136](#)
  - testClassName\_impl, [1140](#)
- CEGUI::TreeEventArgs, [1143](#)

- CEGUI::TreeItem, 1144
  - draw, 1155
  - getFont, 1148
  - getID, 1150
  - getOwnerWindow, 1151
  - getPixelSize, 1154
  - getSelectionBrushImage, 1151
  - getSelectionColours, 1151
  - getText, 1149
  - getTextColours, 1148
  - getUserData, 1150
  - isAutoDeleted, 1150
  - isDisabled, 1150
  - isSelected, 1150
  - setAutoDeleted, 1152
  - setButtonLocation, 1154
  - setDisabled, 1152
  - setFont, 1148
  - setID, 1151
  - setOwnerWindow, 1153
  - setSelected, 1152
  - setSelectionBrushImage, 1154
  - setSelectionColours, 1153, 1154
  - setText, 1151
  - setTextColours, 1149
  - setUserData, 1152
- CEGUI::TreeProperties, 79
- CEGUI::TreeProperties::ForceHorzScrollbar, 342
  - get, 343
  - set, 343
- CEGUI::TreeProperties::ForceVertScrollbar, 358
  - get, 359
  - set, 359
- CEGUI::TreeProperties::ItemTooltips, 511
  - get, 512
  - set, 512
- CEGUI::TreeProperties::MultiSelect, 697
  - get, 698
  - set, 698
- CEGUI::TreeProperties::Sort, 924
  - get, 925
  - set, 925
- CEGUI::UDim, 1156
- CEGUI::UnifiedDim, 1159
  - UnifiedDim, 1160
- CEGUI::UnknownObjectException, 1177
  - UnknownObjectException, 1177
- CEGUI::UpdateEventArgs, 1179
- CEGUI::URect, 1180
- CEGUI::UVector2, 1181
- CEGUI::Vector2, 1186
- CEGUI::Vector3, 1187
- CEGUI::WidgetComponent, 1204
  - findPropertyInitialiser, 1205
  - writeXMLToStream, 1205
- CEGUI::WidgetDim, 1206
  - setSourceDimension, 1207
  - setWidgetName, 1207
  - WidgetDim, 1207
- CEGUI::WidgetLookAndFeel, 1208
  - addImagerySection, 1210
  - addNamedArea, 1213
  - addPropertyDefinition, 1214
  - addPropertyInitialiser, 1211
  - addPropertyLinkDefinition, 1214
  - addStateSpecification, 1211
  - addWidgetComponent, 1211
  - cleanUpWidget, 1212
  - clearImagerySections, 1211
  - clearNamedAreas, 1213
  - clearPropertyDefinitions, 1215
  - clearPropertyInitialisers, 1212
  - clearPropertyLinkDefinitions, 1215
  - clearStateSpecifications, 1212
  - clearWidgetComponents, 1212
  - findPropertyInitialiser, 1215
  - findWidgetComponent, 1216
  - getImagerySection, 1210
  - getName, 1210
  - getNamedArea, 1213
  - getProperties, 1216
  - getPropertyDefinitions, 1216
  - getPropertyLinkDefinitions, 1216
  - getStateImagery, 1210
  - initialiseWidget, 1212
  - isNamedAreaDefined, 1214
  - isStateImageryPresent, 1213
  - layoutChildWidgets, 1214
  - PropertyList, 1210
  - renameChildren, 1215
  - writeXMLToStream, 1215
- CEGUI::WidgetLookManager, 1217
  - addWidgetLook, 1219
  - eraseWidgetLook, 1219
  - getDefaultResourceGroup, 1220
  - getSingleton, 1218
  - getSingletonPtr, 1218
  - getWidgetLook, 1219
  - isWidgetLookAvailable, 1219
  - parseLookNFeelSpecification, 1218
  - setDefaultResourceGroup, 1220
  - writeWidgetLookSeriesToStream, 1220
  - writeWidgetLookToStream, 1220
- CEGUI::Window, 1222
  - activate, 1264
  - addChildWindow, 1266
  - addWindowToDrawList, 1296
  - captureInput, 1268

deactivate, 1264  
destroy, 1272  
disable, 1264  
distributesCapturedInputs, 1257  
doRiseOnClick, 1296  
drawSelf, 1294  
enable, 1263  
EventWindowUpdated, 1297  
getActiveChild, 1250  
getActiveSibling, 1260  
getAlpha, 1252  
getArea, 1279  
getAutoRepeatDelay, 1257  
getAutoRepeatRate, 1257  
getCaptureWindow, 1253  
getChild, 1249  
getChildAtIdx, 1250  
getChildAtPosition, 1254  
getChildCount, 1247  
getChildRecursive, 1249  
getEffectiveAlpha, 1252  
getEventIterator, 1261  
getFont, 1251  
getHeight, 1281  
getHorizontalAlignment, 1259  
getID, 1247  
getInnerRect, 1253  
getLookNFeel, 1259  
getMaxSize, 1281  
getMinSize, 1282  
getModalState, 1259  
getMouseCursor, 1255  
getName, 1245  
getParent, 1255  
getParentPixelHeight, 1261  
getParentPixelSize, 1260  
getParentPixelWidth, 1261  
getPixelRect, 1252  
getPixelRect\_impl, 1252  
getPixelSize, 1255  
getPosition, 1279  
getPrefix, 1245  
getPropertyIterator, 1262  
getRenderCache, 1259  
getSize, 1280  
getTargetChildAtPosition, 1255  
getText, 1251  
getTooltip, 1257  
getTooltipText, 1258  
getTooltipType, 1258  
getType, 1245  
getUnclippedInnerRect, 1253  
getUnclippedInnerRect\_impl, 1253  
getUnclippedPixelRect, 1253  
getUserData, 1256  
getUserString, 1260  
getVerticalAlignment, 1259  
getWidth, 1281  
getWindowRenderer, 1283  
getWindowRendererName, 1283  
getXPosition, 1280  
getYPosition, 1280  
hide, 1264  
inheritsAlpha, 1251  
inheritsTooltipText, 1258  
initialiseComponents, 1262  
isActive, 1247  
isAlwaysOnTop, 1246  
isAncestor, 1250, 1251  
isCapturedByAncestor, 1254  
isCapturedByChild, 1254  
isCapturedByThis, 1254  
isChild, 1247, 1248  
isChildRecursive, 1248  
isClippedByParent, 1247  
isDestroyedByParent, 1246  
isDisabled, 1246  
isDragDropTarget, 1262  
isHit, 1254  
isMouseAutoRepeatEnabled, 1256  
isMousePassThroughEnabled, 1261  
isRiseOnClickEnabled, 1258  
isTopOfZOrder, 1297  
isUserStringDefined, 1260  
isUsingDefaultTooltip, 1257  
isVisible, 1246  
isZOrderingEnabled, 1256  
moveToBack, 1268  
moveToFront, 1267  
moveToFront\_impl, 1296  
onActivated, 1288  
onAlphaChanged, 1285  
onAlwaysOnTopChanged, 1287  
onCaptureGained, 1287  
onCaptureLost, 1287  
onCharacter, 1292  
onChildAdded, 1289  
onChildRemoved, 1289  
onClippingChanged, 1286  
onDeactivated, 1288  
onDestructionStarted, 1288  
onDisabled, 1286  
onDragDropItemDropped, 1293  
onDragDropItemEnters, 1292  
onDragDropItemLeaves, 1292  
onEnabled, 1286  
onFontChanged, 1285  
onHidden, 1285

- onHorizontalAlignmentChanged, 1293
- onIDChanged, 1285
- onInheritsAlphaChanged, 1286
- onKeyDown, 1291
- onKeyUp, 1292
- onMouseButtonDown, 1290
- onMouseButtonUp, 1290
- onMouseClicked, 1291
- onMouseDoubleClicked, 1291
- onMouseEnters, 1289
- onMouseLeaves, 1289
- onMouseMove, 1290
- onMouseTripleClicked, 1291
- onMouseWheel, 1290
- onMoved, 1284
- onParentDestroyChanged, 1286
- onParentSized, 1289
- onRenderingEnded, 1287
- onRenderingStarted, 1287
- onShown, 1285
- onSized, 1284
- onTextChanged, 1284
- onVerticalAlignmentChanged, 1293
- onWindowRendererAttached, 1293
- onWindowRendererDetached, 1294
- onZChanged, 1288
- performChildWindowLayout, 1275
- populateRenderCache, 1294
- releaseInput, 1268
- removeChildWindow, 1267
- removeWindowFromDrawList, 1297
- rename, 1262
- render, 1282
- requestRedraw, 1269
- restoresOldCapture, 1256
- setAlpha, 1269
- setAlwaysOnTop, 1263
- setArea, 1276
- setArea\_impl, 1296
- setAutoRepeatDelay, 1271
- setAutoRepeatRate, 1272
- setClippedByParent, 1265
- setDestroyedByParent, 1263
- setDistributesCapturedInputs, 1272
- setDragDropTarget, 1284
- setEnabled, 1263
- setFalagardType, 1283
- setFont, 1265, 1266
- setHeight, 1278
- setHorizontalAlignment, 1274
- setID, 1265
- setInheritsAlpha, 1269
- setInheritsTooltipText, 1273
- setLookNFeel, 1274
- setMaxSize, 1279
- setMinSize, 1279
- setModalState, 1275
- setMouseAutoRepeatEnabled, 1271
- setMouseCursor, 1270
- setMousePassThroughEnabled, 1283
- setParent, 1295
- setPosition, 1277
- setPrefix, 1265
- setRestoreCapture, 1268
- setRiseOnClickEnabled, 1274
- setSize, 1278
- setText, 1265
- setTooltip, 1272
- setTooltipText, 1273
- setTooltipType, 1273
- setUserData, 1270
- setUserString, 1275
- setVerticalAlignment, 1274
- setVisible, 1264
- setWantsMultiClickEvents, 1271
- setWidth, 1278
- setWindowRenderer, 1283
- setXPosition, 1277
- setYPosition, 1277
- setZOrderingEnabled, 1271
- show, 1264
- testClassName, 1258
- testClassName\_impl, 1295
- update, 1282
- updateSelf, 1294
- validateWindowRenderer, 1295
- wantsMultiClickEvents, 1256
- Window, 1245
- writeXMLToStream, 1282
- CEGUI::WindowEventArgs, 1298
- CEGUI::WindowFactory, 1299
  - createWindow, 1299
  - destroyWindow, 1300
  - getTypeName, 1300
- CEGUI::WindowFactoryManager, 1301
  - addFactory, 1303
  - addFalagardWindowMapping, 1305
  - addWindowTypeAlias, 1304
  - getDereferencedAliasType, 1307
  - getFactory, 1304
  - getFalagardMappingForType, 1307
  - getMappedLookForType, 1306
  - getMappedRendererForType, 1306
  - isFactoryPresent, 1304
  - isFalagardMappedType, 1306
  - removeAllFactories, 1303
  - removeFactory, 1303
  - removeFalagardWindowMapping, 1306



- removeWindowTypeAlias, 1305
- CEGUI::WindowFactoryManager::AliasTargetStack, 1308
  - getActiveTarget, 1308
  - getStackedTargetCount, 1308
- CEGUI::WindowFactoryManager::FalagardWindowManager, 1309
- CEGUI::WindowManager, 1310
  - ~WindowManager, 1312
  - cleanDeadPool, 1315
  - createWindow, 1312
  - DEBUG\_dumpWindowNames, 1317
  - destroyAllWindows, 1314
  - destroyWindow, 1313
  - getDefaultResourceGroup, 1316
  - getWindow, 1313
  - isDeadPoolEmpty, 1315
  - isWindowPresent, 1314
  - loadWindowLayout, 1314
  - PropertyCallback, 1312
  - renameWindow, 1316
  - setDefaultResourceGroup, 1317
  - WindowManager, 1312
  - writeWindowLayoutToStream, 1315, 1316
- CEGUI::WindowProperties, 80
- CEGUI::WindowProperties::Alpha, 90
  - get, 91
  - set, 91
- CEGUI::WindowProperties::AlwaysOnTop, 94
  - get, 95
  - set, 95
- CEGUI::WindowProperties::AutoRepeatDelay, 96
  - get, 97
  - set, 97
- CEGUI::WindowProperties::AutoRepeatRate, 98
  - get, 99
  - set, 99
- CEGUI::WindowProperties::ClippedByParent, 128
  - get, 129
  - set, 129
- CEGUI::WindowProperties::CustomTooltipType, 205
  - get, 206
  - set, 206
- CEGUI::WindowProperties::DestroyedByParent, 209
  - get, 210
  - set, 210
- CEGUI::WindowProperties::Disabled, 214
  - get, 215
  - isDefault, 215
  - set, 215
- CEGUI::WindowProperties::DistributeCapturedInputs, 218
  - get, 219
  - set, 219
- CEGUI::WindowProperties::DragDropTarget, 242
  - get, 243
  - set, 243
- CEGUI::WindowProperties::Font, 309
  - get, 310
  - isDefault, 310
  - set, 310
- CEGUI::WindowProperties::HorizontalAlignment, 412
  - get, 413
  - set, 413
- CEGUI::WindowProperties::ID, 428
  - get, 429
  - set, 429
- CEGUI::WindowProperties::InheritsAlpha, 474
  - get, 475
  - set, 475
- CEGUI::WindowProperties::InheritsTooltipText, 476
  - get, 477
  - set, 477
- CEGUI::WindowProperties::LookNFeel, 587
  - get, 588
  - set, 588
  - writeXMLToStream, 588
- CEGUI::WindowProperties::MouseButtonDownAutoRepeat, 622
  - get, 623
  - set, 623
- CEGUI::WindowProperties::MouseCursorImage, 632
  - get, 633
  - isDefault, 633
  - set, 633
- CEGUI::WindowProperties::MousePassThroughEnabled, 636
  - get, 637
  - set, 637
- CEGUI::WindowProperties::RestoreOldCapture, 804
  - get, 805
  - set, 805
- CEGUI::WindowProperties::RiseOnClick, 806
  - get, 807
  - set, 807
- CEGUI::WindowProperties::Text, 1082
  - get, 1083
  - set, 1083
- CEGUI::WindowProperties::Tooltip, 1112
  - get, 1113
  - set, 1113



- CEGUI::WindowProperties::UnifiedAreaRect,
  - 1157
  - get, 1158
  - set, 1158
- CEGUI::WindowProperties::UnifiedHeight, 1161
  - get, 1162
  - set, 1162
- CEGUI::WindowProperties::UnifiedMaxSize, 1163
  - get, 1164
  - set, 1164
- CEGUI::WindowProperties::UnifiedMinSize, 1165
  - get, 1166
  - set, 1166
- CEGUI::WindowProperties::UnifiedPosition, 1167
  - get, 1168
  - set, 1168
- CEGUI::WindowProperties::UnifiedSize, 1169
  - get, 1170
  - set, 1170
- CEGUI::WindowProperties::UnifiedWidth, 1171
  - get, 1172
  - set, 1172
- CEGUI::WindowProperties::UnifiedXPosition,
  - 1173
  - get, 1174
  - set, 1174
- CEGUI::WindowProperties::UnifiedYPosition,
  - 1175
  - get, 1176
  - set, 1176
- CEGUI::WindowProperties::VerticalAlignment,
  - 1190
  - get, 1191
  - set, 1191
- CEGUI::WindowProperties::Visible, 1200
  - get, 1201
  - isDefault, 1201
  - set, 1201
- CEGUI::WindowProperties::WantsMultiClickEvents, 1202
  - get, 1203
  - set, 1203
- CEGUI::WindowProperties::WindowRenderer,
  - 1323
  - get, 1324
  - set, 1324
  - writeXMLToStream, 1324
- CEGUI::WindowProperties::ZOrderChangeEnabled,
  - 1339
  - get, 1340
  - set, 1340
- CEGUI::WindowRenderer, 1318
  - registerProperty, 1321
  - render, 1321
  - WindowRenderer, 1321
- CEGUI::WindowRendererFactory, 1325
  - WindowRendererFactory, 1325
- CEGUI::XMLAttributes, 1328
  - add, 1329
  - exists, 1329
  - getCount, 1329
  - getName, 1330
  - getValue, 1330
  - getValueAsBool, 1331
  - getValueAsFloat, 1332
  - getValueAsInteger, 1331
  - getValueAsString, 1331
  - remove, 1329
- CEGUI::XMLParser, 1333
  - cleanup, 1334
  - cleanupImpl, 1335
  - getIdentifierString, 1334
  - initialise, 1334
  - initialiseImpl, 1335
  - parseXMLFile, 1334
- CEGUI::XMLSerializer, 1336
  - attribute, 1337
  - closeTag, 1337
  - getTagCount, 1338
  - openTag, 1337
  - operator bool, 1338
  - operator!, 1338
  - text, 1338
  - XMLSerializer, 1337
- Centred
  - CEGUI, 49
- changePopupMenuItem
  - CEGUI::MenuBase, 612
- cleanDeadPool
  - CEGUI::WindowManager, 1315
- cleanup
  - CEGUI::XMLParser, 1334
- cleanupImpl
  - CEGUI::XMLParser, 1335
- cleanUpWidget
  - CEGUI::WidgetLookAndFeel, 1212
- clear
  - CEGUI::String, 1005
- clearAllSelections
  - CEGUI::Combobox, 167
  - CEGUI::Listbox, 530
  - CEGUI::MultiColumnList, 667
  - CEGUI::Tree, 1136
- clearAllSelections\_impl
  - CEGUI::Listbox, 534
  - CEGUI::MultiColumnList, 670
  - CEGUI::Tree, 1140
- clearFrameComponents

- CEGUI::ImagerySection, 451
- clearImageryComponents
  - CEGUI::ImagerySection, 450
- clearImagerySections
  - CEGUI::WidgetLookFeel, 1211
- clearLayers
  - CEGUI::StateImagery, 960
- clearNamedAreas
  - CEGUI::WidgetLookFeel, 1213
- clearProperties
  - CEGUI::PropertySet, 764
- clearPropertyDefinitions
  - CEGUI::WidgetLookFeel, 1215
- clearPropertyInitialisers
  - CEGUI::WidgetLookFeel, 1212
- clearPropertyLinkDefinitions
  - CEGUI::WidgetLookFeel, 1215
- clearRenderList
  - CEGUI::Renderer, 794
- clearSectionSpecifications
  - CEGUI::LayerSpecification, 519
- clearSelection
  - CEGUI::Editbox, 263
- clearStateSpecifications
  - CEGUI::WidgetLookFeel, 1212
- clearTextComponents
  - CEGUI::ImagerySection, 451
- clearWidgetComponents
  - CEGUI::WidgetLookFeel, 1212
- clone
  - CEGUI::BaseDim, 103
- closeAllMenuItemPopups
  - CEGUI::MenuItem, 618
- closePopupMenu
  - CEGUI::MenuItem, 616
  - CEGUI::PopupMenu, 731
- closeTag
  - CEGUI::XMLSerializer, 1337
- compare
  - CEGUI::String, 983–988
- configureScrollbars
  - CEGUI::Listbox, 533
  - CEGUI::MultiColumnList, 670
  - CEGUI::ScrollablePane, 842
- connected
  - CEGUI::BoundSlot, 107
- ConstBaseIterator
  - CEGUI::ConstBaseIterator, 183
- constrainSize
  - CEGUI::Rect, 784
- constrainSizeMax
  - CEGUI::Rect, 784
- constrainSizeMin
  - CEGUI::Rect, 784
- Control
  - CEGUI, 49
- copy
  - CEGUI::String, 990
- createBindings
  - CEGUI::ScriptModule, 827
- createFont
  - CEGUI::FontManager, 332, 333
- createHorzScrollbar
  - CEGUI::Tree, 1139
- createImageset
  - CEGUI::ImagesetManager, 469
- createImagesetFromImageFile
  - CEGUI::ImagesetManager, 469
- createInitialisedSegment
  - CEGUI::ListHeader, 568
- createNewSegment
  - CEGUI::ListHeader, 569
  - CEGUI::ListHeaderWindowRenderer, 583
- createTabButton
  - CEGUI::TabControl, 1071
  - CEGUI::TabControlWindowRenderer, 1075
- createTexture
  - CEGUI::Renderer, 794, 795
- createVertScrollbar
  - CEGUI::Tree, 1139
- createWindow
  - CEGUI::WindowFactory, 1299
  - CEGUI::WindowManager, 1312
- d\_glyphPageLoaded
  - CEGUI::Font, 325
- d\_listItems
  - CEGUI::ItemListBase, 500
- d\_ownerList
  - CEGUI::ItemEntry, 485
- data
  - CEGUI::String, 990
- deactivate
  - CEGUI::Window, 1264
- DEBUG\_dumpWindowNames
  - CEGUI::WindowManager, 1317
- DefaultMouseCursor
  - CEGUI, 48
- defineImage
  - CEGUI::Imageset, 461
- Descending
  - CEGUI::ListHeaderSegment, 576
- destroy
  - CEGUI::ScrollablePane, 842
  - CEGUI::Window, 1272
- destroyAllFonts
  - CEGUI::FontManager, 333
- destroyAllImagesets

- CEGUI::ImagesetManager, 470
- destroyAllTextures
  - CEGUI::Renderer, 795
- destroyAllWindows
  - CEGUI::WindowManager, 1314
- destroyBindings
  - CEGUI::ScriptModule, 828
- destroyFont
  - CEGUI::FontManager, 333
- destroyImageset
  - CEGUI::ImagesetManager, 470
- destroyListSegment
  - CEGUI::ListHeader, 569
  - CEGUI::ListHeaderWindowRenderer, 583
- destroyTexture
  - CEGUI::Renderer, 795
- destroyWindow
  - CEGUI::WindowFactory, 1300
  - CEGUI::WindowManager, 1313
- Dimension
  - CEGUI::Dimension, 212
- DimensionOperator
  - CEGUI, 46
- DimensionType
  - CEGUI, 46
- disable
  - CEGUI::Window, 1264
- disconnect
  - CEGUI::BoundSlot, 107
- distributesCapturedInputs
  - CEGUI::Window, 1257
- doDragging
  - CEGUI::DragContainer, 233
- doDragMoving
  - CEGUI::ListHeaderSegment, 578
- doDragSizing
  - CEGUI::ListHeaderSegment, 578
- DOP\_ADD
  - CEGUI, 46
- DOP\_DIVIDE
  - CEGUI, 46
- DOP\_MULTIPLY
  - CEGUI, 46
- DOP\_NOOP
  - CEGUI, 46
- DOP\_SUBTRACT
  - CEGUI, 46
- doRender
  - CEGUI::Renderer, 793
- doRiseOnClick
  - CEGUI::Window, 1296
- draw
  - CEGUI::Image, 434–437
  - CEGUI::Imageset, 461, 462
  - CEGUI::ListboxItem, 545
  - CEGUI::ListboxTextItem, 549
  - CEGUI::MouseCursor, 628
  - CEGUI::TreeItem, 1155
- drawGlyphToBuffer
  - CEGUI::FreeTypeFont, 389
- drawSelf
  - CEGUI::ClippedContainer, 134
  - CEGUI::ScrolledContainer, 866
  - CEGUI::TabControl, 1069
  - CEGUI::Window, 1294
- drawText
  - CEGUI::Font, 318–321
- DT\_BOTTOM\_EDGE
  - CEGUI, 46
- DT\_HEIGHT
  - CEGUI, 46
- DT\_INVALID
  - CEGUI, 46
- DT\_LEFT\_EDGE
  - CEGUI, 46
- DT\_RIGHT\_EDGE
  - CEGUI, 46
- DT\_TOP\_EDGE
  - CEGUI, 46
- DT\_WIDTH
  - CEGUI, 46
- DT\_X\_OFFSET
  - CEGUI, 46
- DT\_X\_POSITION
  - CEGUI, 46
- DT\_Y\_OFFSET
  - CEGUI, 46
- DT\_Y\_POSITION
  - CEGUI, 46
- DynamicModule
  - CEGUI::DynamicModule, 252
- empty
  - CEGUI::String, 982
- enable
  - CEGUI::Window, 1263
- end
  - CEGUI::String, 1033
- ensureItemIsVisible
  - CEGUI::Listbox, 532
  - CEGUI::Tree, 1138
- erase
  - CEGUI::String, 1005, 1006
- eraseSelectedText
  - CEGUI::Editbox, 263
  - CEGUI::MultiLineEditbox, 689
- eraseWidgetLook
  - CEGUI::WidgetLookManager, 1219

- Errors
  - CEGUI, [48](#)
- EventDisplaySizeChanged
  - CEGUI::Renderer, [798](#)
- EventWindowUpdated
  - CEGUI::Window, [1297](#)
- Exception
  - CEGUI::Exception, [288](#)
- executeScriptedEventHandler
  - CEGUI::ScriptModule, [827](#)
- executeScriptFile
  - CEGUI::ScriptModule, [826](#)
  - CEGUI::System, [1051](#)
- executeScriptGlobal
  - CEGUI::ScriptModule, [827](#)
  - CEGUI::System, [1051](#)
- executeScriptString
  - CEGUI::System, [1051](#)
- executeString
  - CEGUI::ScriptModule, [827](#)
- exists
  - CEGUI::XMLAttributes, [1329](#)
- FactoryModule
  - CEGUI::FactoryModule, [290](#)
- FadeIn
  - CEGUI::Tooltip, [1117](#)
- FadeOut
  - CEGUI::Tooltip, [1117](#)
- FIC\_BACKGROUND
  - CEGUI, [47](#)
- FIC\_BOTTOM\_EDGE
  - CEGUI, [47](#)
- FIC\_BOTTOM\_LEFT\_CORNER
  - CEGUI, [47](#)
- FIC\_BOTTOM\_RIGHT\_CORNER
  - CEGUI, [47](#)
- FIC\_FRAME\_IMAGE\_COUNT
  - CEGUI, [47](#)
- FIC\_LEFT\_EDGE
  - CEGUI, [47](#)
- FIC\_RIGHT\_EDGE
  - CEGUI, [47](#)
- FIC\_TOP\_EDGE
  - CEGUI, [47](#)
- FIC\_TOP\_LEFT\_CORNER
  - CEGUI, [47](#)
- FIC\_TOP\_RIGHT\_CORNER
  - CEGUI, [47](#)
- FileIOException
  - CEGUI::FileIOException, [307](#)
- find
  - CEGUI::String, [1015–1020](#)
- find\_first\_not\_of
  - CEGUI::String, [1021–1026](#)
- find\_first\_of
  - CEGUI::String, [1021–1026](#)
- find\_last\_not\_of
  - CEGUI::String, [1027–1032](#)
- find\_last\_of
  - CEGUI::String, [1027–1032](#)
- findColumnItemWithText
  - CEGUI::MultiColumnList, [654](#)
- findFirstItemWithID
  - CEGUI::Tree, [1133](#)
- findFirstItemWithText
  - CEGUI::Tree, [1133](#)
- findItemWithText
  - CEGUI::Combobox, [163](#)
  - CEGUI::ItemListBase, [495](#)
  - CEGUI::Listbox, [528](#)
- findListItemWithText
  - CEGUI::MultiColumnList, [655](#)
- findPropertyInitialiser
  - CEGUI::WidgetComponent, [1205](#)
  - CEGUI::WidgetLookAndFeel, [1215](#)
- findRowItemWithText
  - CEGUI::MultiColumnList, [655](#)
- findSelectedItem
  - CEGUI::ItemListbox, [508](#)
- findWidgetComponent
  - CEGUI::WidgetLookAndFeel, [1216](#)
- fireEvent
  - CEGUI::EventSet, [283](#)
  - CEGUI::GlobalEventSet, [398](#)
- FlipHorizontal
  - CEGUI, [49](#)
- FlipVertical
  - CEGUI, [49](#)
- FloatingPoint
  - CEGUI::Spinner, [952](#)
- FMT\_BASELINE
  - CEGUI, [47](#)
- FMT\_HORZ\_EXTENT
  - CEGUI, [47](#)
- FMT\_LINE\_SPACING
  - CEGUI, [47](#)
- Font
  - CEGUI::Font, [316](#)
- Font\_xmlHandler
  - CEGUI::Font\_xmlHandler, [326](#)
- FontDim
  - CEGUI::FontDim, [328](#)
- FontMetricType
  - CEGUI, [46](#)
- formatText
  - CEGUI::MultiLineEditbox, [688](#)
- FrameImageComponent

- CEGUI, [47](#)
- GenericException
  - CEGUI::GenericException, [395](#)
- get
  - CEGUI::CheckboxProperties::Selected, [886](#)
  - CEGUI::ComboboxProperties::CaratIndex, [116](#)
  - CEGUI::ComboboxProperties::EditSelectionLength, [271](#)
  - CEGUI::ComboboxProperties::EditSelectionStart, [273](#)
  - CEGUI::ComboboxProperties::ForceHorzScrollbar, [341](#)
  - CEGUI::ComboboxProperties::ForceVertScrollbar, [355](#)
  - CEGUI::ComboboxProperties::MaxEditTextLength, [594](#)
  - CEGUI::ComboboxProperties::ReadOnly, [779](#)
  - CEGUI::ComboboxProperties::SingleClickMode, [903](#)
  - CEGUI::ComboboxProperties::SortList, [941](#)
  - CEGUI::ComboboxProperties::ValidationString, [1183](#)
  - CEGUI::DragContainerProperties::DragAlpha, [225](#)
  - CEGUI::DragContainerProperties::DragCursorImage, [240](#)
  - CEGUI::DragContainerProperties::DraggingEnabled, [247](#)
  - CEGUI::DragContainerProperties::DragThreshold, [251](#)
  - CEGUI::EditboxProperties::ActiveSelectionColour, [87](#)
  - CEGUI::EditboxProperties::CaratIndex, [118](#)
  - CEGUI::EditboxProperties::InactiveSelectionColour, [473](#)
  - CEGUI::EditboxProperties::MaskCodepoint, [590](#)
  - CEGUI::EditboxProperties::MaskText, [592](#)
  - CEGUI::EditboxProperties::MaxTextLength, [602](#)
  - CEGUI::EditboxProperties::NormalTextColour, [712](#)
  - CEGUI::EditboxProperties::ReadOnly, [777](#)
  - CEGUI::EditboxProperties::SelectedTextColour, [888](#)
  - CEGUI::EditboxProperties::SelectionLength, [894](#)
  - CEGUI::EditboxProperties::SelectionStart, [898](#)
  - CEGUI::EditboxProperties::ValidationString, [1185](#)
  - CEGUI::FrameWindowProperties::CloseButtonEnabled, [136](#)
  - CEGUI::FrameWindowProperties::DragMovingEnabled, [249](#)
  - CEGUI::FrameWindowProperties::EWSizingCursorImage, [285](#)
  - CEGUI::FrameWindowProperties::FrameEnabled, [367](#)
  - CEGUI::FrameWindowProperties::NESWSizingCursorImage, [705](#)
  - CEGUI::FrameWindowProperties::NSSizingCursorImage, [713](#)
  - CEGUI::FrameWindowProperties::NWSEizingCursorImage, [717](#)
  - CEGUI::FrameWindowProperties::RollUpEnabled, [809](#)
  - CEGUI::FrameWindowProperties::RollUpState, [811](#)
  - CEGUI::FrameWindowProperties::SizingBorderThickness, [908](#)
  - CEGUI::FrameWindowProperties::SizingEnabled, [912](#)
  - CEGUI::FrameWindowProperties::TitlebarEnabled, [1111](#)
  - CEGUI::ItemEntryProperties::Selectable, [880](#)
  - CEGUI::ItemEntryProperties::Selected, [884](#)
  - CEGUI::ItemListBaseProperties::AutoResizeEnabled, [101](#)
  - CEGUI::ItemListBaseProperties::SortEnabled, [939](#)
  - CEGUI::ItemListBaseProperties::SortMode, [943](#)
  - CEGUI::ItemListboxProperties::MultiSelect, [700](#)
  - CEGUI::ListboxProperties::ForceHorzScrollbar, [345](#)
  - CEGUI::ListboxProperties::ForceVertScrollbar, [357](#)
  - CEGUI::ListboxProperties::ItemTooltips, [514](#)
  - CEGUI::ListboxProperties::MultiSelect, [702](#)
  - CEGUI::ListboxProperties::Sort, [927](#)
  - CEGUI::ListHeaderProperties::ColumnsMovable, [145](#)
  - CEGUI::ListHeaderProperties::ColumnsSizable, [149](#)
  - CEGUI::ListHeaderProperties::SortColumnID, [931](#)
  - CEGUI::ListHeaderProperties::SortDirection, [935](#)
  - CEGUI::ListHeaderProperties::SortSettingEnabled, [947](#)
  - CEGUI::ListHeaderSegmentProperties::Clickable, [125](#)

- CEGUI::ListHeaderSegmentProperties::Dragable, 223
- CEGUI::ListHeaderSegmentProperties::MovingCursorImage, 638
- CEGUI::ListHeaderSegmentProperties::Sizable, 905
- CEGUI::ListHeaderSegmentProperties::SizingCursorImage, 909
- CEGUI::ListHeaderSegmentProperties::SortDirection, 933
- CEGUI::MenuBaseProperties::AllowMultiplePopups, 89
- CEGUI::MenuBaseProperties::ItemSpacing, 510
- CEGUI::MultiColumnListProperties::ColumnHeader, 143
- CEGUI::MultiColumnListProperties::ColumnsMovable, 147
- CEGUI::MultiColumnListProperties::ColumnsSizable, 151
- CEGUI::MultiColumnListProperties::ForceHorzScrollbar, 347
- CEGUI::MultiColumnListProperties::ForceVertScrollbar, 361
- CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, 708
- CEGUI::MultiColumnListProperties::NominatedSelectionRow, 710
- CEGUI::MultiColumnListProperties::RowCount, 812
- CEGUI::MultiColumnListProperties::SelectionMode, 896
- CEGUI::MultiColumnListProperties::SortColumnID, 929
- CEGUI::MultiColumnListProperties::SortDirection, 937
- CEGUI::MultiColumnListProperties::SortSettingEnabled, 945
- CEGUI::MultiLineEditboxProperties::CaratIndex, 114
- CEGUI::MultiLineEditboxProperties::ForceVertScrollbar, 349
- CEGUI::MultiLineEditboxProperties::MaxTextLength, 600
- CEGUI::MultiLineEditboxProperties::ReadOnly, 781
- CEGUI::MultiLineEditboxProperties::SelectionBrushImage, 889
- CEGUI::MultiLineEditboxProperties::SelectionLength, 892
- CEGUI::MultiLineEditboxProperties::SelectionStart, 900
- CEGUI::MultiLineEditboxProperties::WordWrap, 1327
- CEGUI::PopupMenuProperties::FadeInTime, 293
- CEGUI::PopupMenuProperties::FadeOutTime, 295
- CEGUI::ProgressBarProperties::CurrentProgress, 200
- CEGUI::ProgressBarProperties::StepSize, 963
- CEGUI::Property, 742
- CEGUI::PropertyDefinition, 749
- CEGUI::PropertyLinkDefinition, 760
- CEGUI::RadioButtonProperties::GroupID, 404
- CEGUI::RadioButtonProperties::Selected, 882
- CEGUI::ScrollablePaneProperties::ContentArea, 186
- CEGUI::ScrollablePaneProperties::ContentPaneAutoSized, 190
- CEGUI::ScrollablePaneProperties::ForceHorzScrollbar, 337
- CEGUI::ScrollablePaneProperties::ForceVertScrollbar, 351
- CEGUI::ScrollablePaneProperties::HorzOverlapSize, 417
- CEGUI::ScrollablePaneProperties::HorzScrollPosition, 421
- CEGUI::ScrollablePaneProperties::HorzStepSize, 423
- CEGUI::ScrollablePaneProperties::VertOverlapSize, 1193
- CEGUI::ScrollablePaneProperties::VertScrollPosition, 1197
- CEGUI::ScrollablePaneProperties::VertStepSize, 1199
- CEGUI::ScrollbarProperties::DocumentSize, 221
- CEGUI::ScrollbarProperties::OverlapSize, 722
- CEGUI::ScrollbarProperties::PageSize, 724
- CEGUI::ScrollbarProperties::ScrollPosition, 872
- CEGUI::ScrollbarProperties::StepSize, 967
- CEGUI::ScrolledContainerProperties::ChildExtentsArea, 123
- CEGUI::ScrolledContainerProperties::ContentArea, 188
- CEGUI::ScrolledContainerProperties::ContentPaneAutoSized, 192
- CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar, 339
- CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar, 353
- CEGUI::SliderProperties::ClickStepSize, 127
- CEGUI::SliderProperties::CurrentValue, 202



- CEGUI::SliderProperties::MaximumValue, [596](#)
- CEGUI::SpinnerProperties::CurrentValue, [204](#)
- CEGUI::SpinnerProperties::MaximumValue, [598](#)
- CEGUI::SpinnerProperties::MinimumValue, [621](#)
- CEGUI::SpinnerProperties::StepSize, [965](#)
- CEGUI::SpinnerProperties::TextInputMode, [1090](#)
- CEGUI::TabControlProperties::TabHeight, [1076](#)
- CEGUI::TabControlProperties::TabPagePosition, [1078](#)
- CEGUI::TabControlProperties::TabTextPadding, [1080](#)
- CEGUI::ThumbProperties::HorzFree, [415](#)
- CEGUI::ThumbProperties::HorzRange, [419](#)
- CEGUI::ThumbProperties::HotTracked, [425](#)
- CEGUI::ThumbProperties::VertFree, [1189](#)
- CEGUI::ThumbProperties::VertRange, [1195](#)
- CEGUI::TitlebarProperties::DraggingEnabled, [245](#)
- CEGUI::TooltipProperties::DisplayTime, [217](#)
- CEGUI::TooltipProperties::FadeTime, [297](#)
- CEGUI::TooltipProperties::HoverTime, [427](#)
- CEGUI::TreeProperties::ForceHorzScrollbar, [343](#)
- CEGUI::TreeProperties::ForceVertScrollbar, [359](#)
- CEGUI::TreeProperties::ItemTooltips, [512](#)
- CEGUI::TreeProperties::MultiSelect, [698](#)
- CEGUI::TreeProperties::Sort, [925](#)
- CEGUI::WindowProperties::Alpha, [91](#)
- CEGUI::WindowProperties::AlwaysOnTop, [95](#)
- CEGUI::WindowProperties::AutoRepeatDelay, [97](#)
- CEGUI::WindowProperties::AutoRepeatRate, [99](#)
- CEGUI::WindowProperties::ClippedByParent, [129](#)
- CEGUI::WindowProperties::CustomTooltipType, [206](#)
- CEGUI::WindowProperties::DestroyedByParent, [210](#)
- CEGUI::WindowProperties::Disabled, [215](#)
- CEGUI::WindowProperties::DistributeCapturedInputs, [219](#)
- CEGUI::WindowProperties::DragDropTarget, [243](#)
- CEGUI::WindowProperties::Font, [310](#)
- CEGUI::WindowProperties::HorizontalAlignment, [413](#)
- CEGUI::WindowProperties::ID, [429](#)
- CEGUI::WindowProperties::InheritsAlpha, [475](#)
- CEGUI::WindowProperties::InheritsTooltipText, [477](#)
- CEGUI::WindowProperties::LookNFeel, [588](#)
- CEGUI::WindowProperties::MouseButtonDownAutoRepeat, [623](#)
- CEGUI::WindowProperties::MouseCursorImage, [633](#)
- CEGUI::WindowProperties::MousePassThroughEnabled, [637](#)
- CEGUI::WindowProperties::RestoreOldCapture, [805](#)
- CEGUI::WindowProperties::RiseOnClick, [807](#)
- CEGUI::WindowProperties::Text, [1083](#)
- CEGUI::WindowProperties::Tooltip, [1113](#)
- CEGUI::WindowProperties::UnifiedAreaRect, [1158](#)
- CEGUI::WindowProperties::UnifiedHeight, [1162](#)
- CEGUI::WindowProperties::UnifiedMaxSize, [1164](#)
- CEGUI::WindowProperties::UnifiedMinSize, [1166](#)
- CEGUI::WindowProperties::UnifiedPosition, [1168](#)
- CEGUI::WindowProperties::UnifiedSize, [1170](#)
- CEGUI::WindowProperties::UnifiedWidth, [1172](#)
- CEGUI::WindowProperties::UnifiedXPosition, [1174](#)
- CEGUI::WindowProperties::UnifiedYPosition, [1176](#)
- CEGUI::WindowProperties::VerticalAlignment, [1191](#)
- CEGUI::WindowProperties::Visible, [1201](#)
- CEGUI::WindowProperties::WantsMultiClickEvents, [1203](#)
- CEGUI::WindowProperties::WindowRenderer, [1324](#)
- CEGUI::WindowProperties::ZOrderChangeEnabled, [1340](#)
- CEGUI::Window, [1250](#)
- CEGUI::Window::getActiveSibling, [1260](#)
- CEGUI::Window::getActiveTarget, [1308](#)
- CEGUI::WindowFactoryManager::AliasTargetStack, [1308](#)
- CEGUI::Window::getAdjustDirectionFromPoint, [857](#)
- CEGUI::Scrollbar, [857](#)
- CEGUI::ScrollbarWindowRenderer, [861](#)

- CEGUI::Slider, 917
- CEGUI::SliderWindowRenderer, 922
- getAdvance
  - CEGUI::FontGlyph, 330
- getAlpha
  - CEGUI::Window, 1252
- getArea
  - CEGUI::NamedArea, 703
  - CEGUI::Window, 1279
- getAreaPropertySource
  - CEGUI::ComponentArea, 179
- getAutoRepeatDelay
  - CEGUI::Window, 1257
- getAutoRepeatRate
  - CEGUI::Window, 1257
- getBackgroundHorizontalFormatting
  - CEGUI::FrameComponent, 364
- getBackgroundVerticalFormatting
  - CEGUI::FrameComponent, 363
- getBaseDimension
  - CEGUI::Dimension, 212
- getBaseline
  - CEGUI::Font, 322
- getCaptureWindow
  - CEGUI::Window, 1253
- getCaratIndex
  - CEGUI::Combobox, 161
  - CEGUI::Editbox, 260
  - CEGUI::MultiLineEditbox, 685
- getCharAtPixel
  - CEGUI::Font, 323
- getChild
  - CEGUI::Window, 1249
- getChildAtIndex
  - CEGUI::Window, 1250
- getChildAtPosition
  - CEGUI::Window, 1254
- getChildCount
  - CEGUI::Window, 1247
- getChildExtentsArea
  - CEGUI::ScrolledContainer, 865
- getChildRecursive
  - CEGUI::Window, 1249
- getClickStep
  - CEGUI::Slider, 916
- getClipArea
  - CEGUI::ClippedContainer, 133
- getCloseButton
  - CEGUI::FrameWindow, 381
- getColourAtPoint
  - CEGUI::ColourRect, 141
- getColours
  - CEGUI::FalagardComponentBase, 302
- getColumnCount
  - CEGUI::ListHeader, 559
  - CEGUI::MultiColumnList, 650
- getColumnFromID
  - CEGUI::ListHeader, 560
- getColumnFromSegment
  - CEGUI::ListHeader, 560
- getColumnHeaderWidth
  - CEGUI::MultiColumnList, 651
- getColumnID
  - CEGUI::MultiColumnList, 658
- getColumnWidth
  - CEGUI::ListHeader, 562
- getColumnWithHeaderText
  - CEGUI::MultiColumnList, 651
- getColumnWithID
  - CEGUI::MultiColumnList, 651
- getColumnWithText
  - CEGUI::ListHeader, 561
- getComponentArea
  - CEGUI::FalagardComponentBase, 302
- getConstraintArea
  - CEGUI::MouseCursor, 630
- getContentArea
  - CEGUI::ScrolledContainer, 865
- getContentPane
  - CEGUI::GroupBox, 402
  - CEGUI::ItemListBase, 498
  - CEGUI::ScrollablePane, 837
- getContentPaneArea
  - CEGUI::ScrollablePane, 838
- getContentSize
  - CEGUI::ItemListBase, 499
  - CEGUI::Menubar, 608
  - CEGUI::PopupMenu, 732
- getCount
  - CEGUI::XMLAttributes, 1329
- getCurrentDropTarget
  - CEGUI::DragContainer, 232
- getCurrentValue
  - CEGUI::Slider, 916
  - CEGUI::Spinner, 952
- getCurrentZ
  - CEGUI::Renderer, 797
- getDataPtr
  - CEGUI::RawDataContainer, 774
- getDecreaseButton
  - CEGUI::Scrollbar, 854
  - CEGUI::Spinner, 955
- getDefault
  - CEGUI::Property, 746
- getDefaultFont
  - CEGUI::System, 1046
- getDefaultMouseCursor
  - CEGUI::System, 1049



- getDefaultResourceGroup
  - CEGUI::Font, [324](#)
  - CEGUI::Imageset, [464](#)
  - CEGUI::ResourceProvider, [802](#)
  - CEGUI::Scheme, [816](#)
  - CEGUI::ScriptModule, [829](#)
  - CEGUI::WidgetLookManager, [1220](#)
  - CEGUI::WindowManager, [1316](#)
- getDefaultTooltip
  - CEGUI::System, [1053](#)
- getDefaultXMLParserName
  - CEGUI::System, [1054](#)
- getDereferencedAliasType
  - CEGUI::WindowFactoryManager, [1307](#)
- getDimensionOperator
  - CEGUI::BaseDim, [104](#)
- getDimensionType
  - CEGUI::Dimension, [212](#)
- getDisplayIndependantPosition
  - CEGUI::MouseCursor, [630](#)
- getDisplayTime
  - CEGUI::Tooltip, [1119](#)
- getDocumentSize
  - CEGUI::Scrollbar, [852](#)
- getDragAlpha
  - CEGUI::DragContainer, [231](#)
- getDragCursorImage
  - CEGUI::DragContainer, [231](#)
- getDragMoveOffset
  - CEGUI::ListHeaderSegment, [576](#)
- getDropList
  - CEGUI::Combobox, [160](#)
- getEditbox
  - CEGUI::Combobox, [159](#)
  - CEGUI::Spinner, [956](#)
- getEffectiveAlpha
  - CEGUI::Window, [1252](#)
- getEventIterator
  - CEGUI::Window, [1261](#)
- getEventObject
  - CEGUI::EventSet, [284](#)
- getEWSizingCursorImage
  - CEGUI::FrameWindow, [377](#)
- getFactory
  - CEGUI::WindowFactoryManager, [1304](#)
- getFadeInTime
  - CEGUI::PopupMenu, [730](#)
- getFadeOutTime
  - CEGUI::PopupMenu, [730](#)
- getFadeTime
  - CEGUI::Tooltip, [1118](#)
- getFalagardMappingForType
  - CEGUI::WindowFactoryManager, [1307](#)
- getFileName
  - CEGUI::Exception, [288](#)
- getFirstSelectedItem
  - CEGUI::ItemListBox, [506](#)
  - CEGUI::ListBox, [526](#)
  - CEGUI::MultiColumnList, [656](#)
  - CEGUI::Tree, [1132](#)
- getFont
  - CEGUI::FontManager, [334](#)
  - CEGUI::ListBoxTextItem, [547](#)
  - CEGUI::TextComponent, [1086](#)
  - CEGUI::TreeItem, [1148](#)
  - CEGUI::Window, [1251](#)
- getFontHeight
  - CEGUI::Font, [322](#)
- getFontPropertySource
  - CEGUI::TextComponent, [1088](#)
- getFormattedLineCount
  - CEGUI::Font, [323](#)
- getFormattedTextExtent
  - CEGUI::Font, [324](#)
- getGlyphData
  - CEGUI::Font, [316](#)
- getGroupID
  - CEGUI::RadioButton, [771](#)
- getGUISheet
  - CEGUI::System, [1047](#)
- getHeaderSegmentForColumn
  - CEGUI::MultiColumnList, [652](#)
- getHeight
  - CEGUI::Image, [432](#)
  - CEGUI::Renderer, [796](#)
  - CEGUI::Texture, [1093](#)
  - CEGUI::Window, [1281](#)
- getHeightToItemInList
  - CEGUI::Tree, [1140](#)
- getHelp
  - CEGUI::Property, [742](#)
- getHorizontalAlignment
  - CEGUI::Window, [1259](#)
- getHorizontalFormatting
  - CEGUI::ImageryComponent, [446](#)
  - CEGUI::TextComponent, [1087](#)
- getHorizontalOverlapSize
  - CEGUI::ScrollablePane, [839](#)
- getHorizontalScrollPosition
  - CEGUI::ScrollablePane, [839](#)
- getHorizontalStepSize
  - CEGUI::ScrollablePane, [839](#)
- getHorzRange
  - CEGUI::Thumb, [1101](#)
- getHorzScreenDPI
  - CEGUI::Renderer, [796](#)
- getHorzScrollbar
  - CEGUI::ListBox, [533](#)

- CEGUI::MultiColumnList, 659
- CEGUI::MultiLineEditbox, 686
- CEGUI::ScrollablePane, 841
- getHoverTime
  - CEGUI::Tooltip, 1118
- getID
  - CEGUI::ListboxItem, 540
  - CEGUI::TreeItem, 1150
  - CEGUI::Window, 1247
- getIdentifierString
  - CEGUI::ImageCodec, 439
  - CEGUI::Renderer, 797
  - CEGUI::ScriptModule, 828
  - CEGUI::XMLParser, 1334
- getImage
  - CEGUI::FrameComponent, 364
  - CEGUI::ImageryComponent, 445
  - CEGUI::Imageset, 458
  - CEGUI::MouseCursor, 627
- getImageCount
  - CEGUI::Imageset, 458
- getImageHeight
  - CEGUI::Imageset, 459
- getImageOffset
  - CEGUI::Imageset, 460
- getImageOffsetX
  - CEGUI::Imageset, 460
- getImageOffsetY
  - CEGUI::Imageset, 460
- getImagePropertySource
  - CEGUI::ImageryComponent, 446
- getImagerySection
  - CEGUI::WidgetLookAndFeel, 1210
- getImageset
  - CEGUI::Image, 433
  - CEGUI::ImagesetManager, 470
- getImagesetName
  - CEGUI::Image, 433
- getImageSize
  - CEGUI::Imageset, 459
- getImageWidth
  - CEGUI::Imageset, 459
- getIncreaseButton
  - CEGUI::Scrollbar, 854
  - CEGUI::Spinner, 955
- getInitialiserValue
  - CEGUI::PropertyInitialiser, 758
- getInnerRect
  - CEGUI::Window, 1253
- getIntersection
  - CEGUI::Rect, 783
- getItemAtGridReference
  - CEGUI::MultiColumnList, 653
- getItemAtPoint
  - CEGUI::Listbox, 534
  - CEGUI::MultiColumnList, 671
  - CEGUI::Tree, 1140
- getItemColumnIndex
  - CEGUI::MultiColumnList, 652
- getItemCount
  - CEGUI::Combobox, 162
  - CEGUI::ItemListBase, 494
  - CEGUI::Listbox, 526
  - CEGUI::Tree, 1132
- getItemFromIndex
  - CEGUI::ItemListBase, 494
- getItemGridReference
  - CEGUI::MultiColumnList, 653
- getItemIndex
  - CEGUI::Combobox, 162
  - CEGUI::ItemListBase, 494
  - CEGUI::Listbox, 527
- getItemPixelSize
  - CEGUI::ItemEntry, 483
  - CEGUI::ItemEntryWindowRenderer, 488
- getItemRenderArea
  - CEGUI::ItemListBase, 498
  - CEGUI::ItemListBaseWindowRenderer, 503
- getItemRowIndex
  - CEGUI::MultiColumnList, 652
- getItemSpacing
  - CEGUI::MenuBase, 611
- getLastSelectedItem
  - CEGUI::ItemListbox, 506
  - CEGUI::Tree, 1132
- getLayerPriority
  - CEGUI::LayerSpecification, 519
- getLine
  - CEGUI::Exception, 289
- getLineSpacing
  - CEGUI::Font, 322
- getListboxItemFromIndex
  - CEGUI::Combobox, 162
  - CEGUI::Listbox, 527
- getListHeader
  - CEGUI::MultiColumnList, 659
- getListRenderArea
  - CEGUI::Listbox, 533
  - CEGUI::ListboxWindowRenderer, 552
  - CEGUI::MultiColumnList, 659
  - CEGUI::MultiColumnListWindowRenderer, 677
- getLoggingLevel
  - CEGUI::Logger, 585
- getLookNFeel
  - CEGUI::Window, 1259
- getMappedLookForType
  - CEGUI::WindowFactoryManager, 1306

- getMappedRendererForType
  - CEGUI::WindowFactoryManager, 1306
- getMaskCodePoint
  - CEGUI::Editbox, 261
- getMasterColours
  - CEGUI::ImagerySection, 451
- getMaximumValue
  - CEGUI::Spinner, 953
- getMaxSize
  - CEGUI::Window, 1281
- getMaxTextLength
  - CEGUI::Combobox, 161
  - CEGUI::Editbox, 261
  - CEGUI::MultiLineEditbox, 685
- getMaxTextureSize
  - CEGUI::Renderer, 796
- getMaxValue
  - CEGUI::Slider, 916
- getMessage
  - CEGUI::Exception, 288
- getMinimumValue
  - CEGUI::Spinner, 953
- getMinSize
  - CEGUI::Window, 1282
- getModalState
  - CEGUI::Window, 1259
- getModalTarget
  - CEGUI::System, 1053
- getMouseCursor
  - CEGUI::Window, 1255
- getMouseMoveScaling
  - CEGUI::System, 1052
- getMultiClickTimeout
  - CEGUI::System, 1048
- getMultiClickToleranceAreaSize
  - CEGUI::System, 1048
- getName
  - CEGUI::Event, 275
  - CEGUI::Exception, 288
  - CEGUI::Image, 433
  - CEGUI::ImagerySection, 452
  - CEGUI::Imageset, 458
  - CEGUI::NamedArea, 703
  - CEGUI::Property, 742
  - CEGUI::Scheme, 816
  - CEGUI::StateImagery, 961
  - CEGUI::WidgetLookFeel, 1210
  - CEGUI::Window, 1245
  - CEGUI::XMLAttributes, 1330
- getNamedArea
  - CEGUI::WidgetLookFeel, 1213
- getNativeResolution
  - CEGUI::Imageset, 463
- getNESWSizingCursorImage
  - CEGUI::FrameWindow, 378
- getNextSelected
  - CEGUI::Listbox, 526
  - CEGUI::MultiColumnList, 656
  - CEGUI::Tree, 1132
- getNextSelectedItem
  - CEGUI::ItemListbox, 507
- getNextSelectedItemAfter
  - CEGUI::ItemListbox, 507
- getNextTokenLength
  - CEGUI::MultiLineEditbox, 689
- getNextWord
  - CEGUI::TextUtils, 1096
- getNextWordStartIdx
  - CEGUI::TextUtils, 1096
- getNominatedSelectionColumn
  - CEGUI::MultiColumnList, 657
- getNominatedSelectionColumnID
  - CEGUI::MultiColumnList, 657
- getNominatedSelectionRow
  - CEGUI::MultiColumnList, 657
- getNSSizingCursorImage
  - CEGUI::FrameWindow, 377
- getNWSEsizingCursorImage
  - CEGUI::FrameWindow, 378
- getOffsets
  - CEGUI::Image, 433
- getOffsetX
  - CEGUI::Image, 433
- getOffsetY
  - CEGUI::Image, 433
- getOperand
  - CEGUI::BaseDim, 104
- getOriginalHeight
  - CEGUI::Texture, 1093
- getOriginalWidth
  - CEGUI::Texture, 1092
- getOverlapSize
  - CEGUI::Scrollbar, 853
- getOverrideColours
  - CEGUI::SectionSpecification, 876
- getOwnerWidgetLookFeel
  - CEGUI::SectionSpecification, 875
- getOwnerWindow
  - CEGUI::ListboxItem, 541
  - CEGUI::TreeItem, 1151
- getPageSize
  - CEGUI::Scrollbar, 853
- getParent
  - CEGUI::Window, 1255
- getParentPixelHeight
  - CEGUI::Window, 1261
- getParentPixelSize
  - CEGUI::Window, 1260

- getParentPixelWidth
  - CEGUI::Window, 1261
- getPixelDragThreshold
  - CEGUI::DragContainer, 230
- getPixelOffsetToColumn
  - CEGUI::ListHeader, 561
- getPixelOffsetToSegment
  - CEGUI::ListHeader, 561
- getPixelRect
  - CEGUI::ComponentArea, 178
  - CEGUI::Window, 1252
- getPixelRect\_impl
  - CEGUI::Window, 1252
- getPixelSize
  - CEGUI::ListboxItem, 545
  - CEGUI::ListboxTextItem, 549
  - CEGUI::TreeItem, 1154
  - CEGUI::Window, 1255
- getPopupMenu
  - CEGUI::MenuItem, 616
- getPopupMenuItem
  - CEGUI::MenuBase, 612
- getPosition
  - CEGUI::MouseCursor, 629
  - CEGUI::Window, 1279
- getPrefix
  - CEGUI::Window, 1245
- getProperties
  - CEGUI::WidgetLookFeel, 1216
- getProperty
  - CEGUI::PropertySet, 765
- getPropertyDefault
  - CEGUI::PropertySet, 766
- getPropertyDefinitions
  - CEGUI::WidgetLookFeel, 1216
- getPropertyHelp
  - CEGUI::PropertySet, 765
- getPropertyIterator
  - CEGUI::Window, 1262
- getPropertyLinkDefinitions
  - CEGUI::WidgetLookFeel, 1216
- getPushButton
  - CEGUI::Combobox, 159
- getRect
  - CEGUI::Renderer, 796
- getRenderCache
  - CEGUI::Window, 1259
- getRenderedAdvance
  - CEGUI::FontGlyph, 330
- getRenderer
  - CEGUI::System, 1046
  - CEGUI::Texture, 1094
- getResourceProvider
  - CEGUI::System, 1051
- getRowCount
  - CEGUI::MultiColumnList, 650
- getRowID
  - CEGUI::MultiColumnList, 658
- getRowWithID
  - CEGUI::MultiColumnList, 659
- getScheme
  - CEGUI::SchemeManager, 820
- getScriptingModule
  - CEGUI::System, 1050
- getScrolledContainer
  - CEGUI::ScrollablePane, 843
- getScrollPosition
  - CEGUI::Scrollbar, 854
- getSectionName
  - CEGUI::SectionSpecification, 875
- getSegmentFromColumn
  - CEGUI::ListHeader, 559
- getSegmentFromID
  - CEGUI::ListHeader, 559
- getSegmentOffset
  - CEGUI::ListHeader, 563
- getSelectedButtonInGroup
  - CEGUI::RadioButton, 772
- getSelectedCount
  - CEGUI::Listbox, 526
  - CEGUI::MultiColumnList, 656
  - CEGUI::Tree, 1132
- getSelectedItem
  - CEGUI::Combobox, 162
- getSelectedTabIndex
  - CEGUI::TabControl, 1068
- getSelectionBrushImage
  - CEGUI::ListboxItem, 542
  - CEGUI::TreeItem, 1151
- getSelectionColours
  - CEGUI::ListboxItem, 541
  - CEGUI::TreeItem, 1151
- getSelectionEndIndex
  - CEGUI::Combobox, 161
  - CEGUI::Editbox, 260
  - CEGUI::MultiLineEditbox, 685
- getSelectionLength
  - CEGUI::Combobox, 161
  - CEGUI::Editbox, 261
  - CEGUI::MultiLineEditbox, 685
- getSelectionMode
  - CEGUI::MultiColumnList, 657
- getSelectionStartIndex
  - CEGUI::Combobox, 161
  - CEGUI::Editbox, 260
  - CEGUI::MultiLineEditbox, 685
- getSingleClickEnabled
  - CEGUI::Combobox, 159

- getSingleClickTimeout
  - CEGUI::System, 1048
- getSingleton
  - CEGUI::GlobalEventSet, 397
  - CEGUI::MouseCursor, 627
  - CEGUI::System, 1046
  - CEGUI::WidgetLookManager, 1218
- getSingletonPtr
  - CEGUI::GlobalEventSet, 398
  - CEGUI::MouseCursor, 627
  - CEGUI::System, 1046
  - CEGUI::WidgetLookManager, 1218
- getSize
  - CEGUI::Image, 432
  - CEGUI::RawDataContainer, 775
  - CEGUI::Renderer, 796
  - CEGUI::Window, 1280
- getSizingBorderAtPoint
  - CEGUI::FrameWindow, 382
- getSizingBorderThickness
  - CEGUI::FrameWindow, 375
- getSortColumn
  - CEGUI::ListHeader, 560
  - CEGUI::MultiColumnList, 650
- getSortDirection
  - CEGUI::ListHeader, 562
  - CEGUI::ListHeaderSegment, 576
  - CEGUI::MultiColumnList, 652
- getSortSegment
  - CEGUI::ListHeader, 559
- getSourceTextureArea
  - CEGUI::Image, 434
- getStackedTargetCount
  - CEGUI::WindowFactoryManager::AliasTargetStack, 1308
- getStateImagery
  - CEGUI::WidgetLookFeel, 1210
- getStepSize
  - CEGUI::Scrollbar, 853
  - CEGUI::Spinner, 952
- getSubRectangle
  - CEGUI::ColourRect, 140
- getSupportedFormat
  - CEGUI::ImageCodec, 440
- getSymbolAddress
  - CEGUI::DynamicModule, 253
- getSystemKeys
  - CEGUI::System, 1052
- getTabButtonPane
  - CEGUI::TabControl, 1070
- getTabContents
  - CEGUI::TabControl, 1067, 1068
- getTabContentsAtIndex
  - CEGUI::TabControl, 1067
- getTabCount
  - CEGUI::TabControl, 1067
- getTabPane
  - CEGUI::TabControl, 1070
- getTabPanePosition
  - CEGUI::TabControl, 1067
- getTagCount
  - CEGUI::XMLSerializer, 1338
- getTargetChildAtPosition
  - CEGUI::Window, 1255
- getTargetPropertyName
  - CEGUI::PropertyInitialiser, 758
- getTargetWindow
  - CEGUI::PropertyLinkDefinition, 761
  - CEGUI::Tooltip, 1118
- getText
  - CEGUI::ListboxItem, 540
  - CEGUI::TextComponent, 1085
  - CEGUI::TreeItem, 1149
  - CEGUI::Window, 1251
- getTextColours
  - CEGUI::ListboxTextItem, 547
  - CEGUI::TreeItem, 1148
- getTextExtent
  - CEGUI::Font, 322
- getTextFromValue
  - CEGUI::Spinner, 955
- getTextIndexFromPosition
  - CEGUI::Editbox, 263
  - CEGUI::EditboxWindowRenderer, 269
  - CEGUI::MultiLineEditbox, 689
- getTextInputMode
  - CEGUI::Spinner, 953
- getTextPropertySource
  - CEGUI::TextComponent, 1087
- getTextRenderArea
  - CEGUI::MultiLineEditbox, 686
  - CEGUI::MultiLineEditboxWindowRenderer, 696
- getTextSize
  - CEGUI::Tooltip, 1120
  - CEGUI::TooltipWindowRenderer, 1125
- getTextSize\_impl
  - CEGUI::Tooltip, 1120
- getTexture
  - CEGUI::Imageset, 458
- getTextureSize
  - CEGUI::FreeTypeFont, 389
- getThumb
  - CEGUI::Scrollbar, 854
  - CEGUI::Slider, 916
- getTitlebar
  - CEGUI::FrameWindow, 381
- getTooltip

- CEGUI::Window, 1257
- getTooltipText
  - CEGUI::Window, 1258
- getTooltipType
  - CEGUI::Window, 1258
- getTotalColumnHeadersWidth
  - CEGUI::MultiColumnList, 651
- getTotalSegmentsPixelExtent
  - CEGUI::ListHeader, 561
- getTreeRenderArea
  - CEGUI::Tree, 1139
- getType
  - CEGUI::Window, 1245
- getTypeName
  - CEGUI::WindowFactory, 1300
- getUnclippedInnerRect
  - CEGUI::Window, 1253
- getUnclippedInnerRect\_impl
  - CEGUI::ClippedContainer, 133
  - CEGUI::ScrolledContainer, 865
  - CEGUI::Window, 1253
- getUnclippedPixelRect
  - CEGUI::Window, 1253
- getUnifiedConstraintArea
  - CEGUI::MouseCursor, 630
- getUserData
  - CEGUI::ListboxItem, 540
  - CEGUI::TreeItem, 1150
  - CEGUI::Window, 1256
- getUserString
  - CEGUI::Window, 1260
- getValidationString
  - CEGUI::Combobox, 161
  - CEGUI::Editbox, 260
- getValue
  - CEGUI::BaseDim, 103
  - CEGUI::XMLAttributes, 1330
- getValueAsBool
  - CEGUI::XMLAttributes, 1331
- getValueAsFloat
  - CEGUI::XMLAttributes, 1332
- getValueAsInteger
  - CEGUI::XMLAttributes, 1331
- getValueAsString
  - CEGUI::XMLAttributes, 1331
- getValueFromText
  - CEGUI::Spinner, 954
- getValueFromThumb
  - CEGUI::Scrollbar, 857
  - CEGUI::ScrollbarWindowRenderer, 861
  - CEGUI::Slider, 917
  - CEGUI::SliderWindowRenderer, 922
- getVerticalAlignment
  - CEGUI::Window, 1259
- getVerticalFormatting
  - CEGUI::ImageryComponent, 445
  - CEGUI::TextComponent, 1086
- getVerticalOverlapSize
  - CEGUI::ScrollablePane, 840
- getVerticalScrollPosition
  - CEGUI::ScrollablePane, 841
- getVerticalStepSize
  - CEGUI::ScrollablePane, 840
- getVertRange
  - CEGUI::Thumb, 1101
- getVertScreenDPI
  - CEGUI::Renderer, 797
- getVertScrollbar
  - CEGUI::Listbox, 533
  - CEGUI::MultiColumnList, 659
  - CEGUI::MultiLineEditbox, 686
  - CEGUI::ScrollablePane, 841
- getViewableArea
  - CEGUI::ScrollablePane, 841
  - CEGUI::ScrollablePaneWindowRenderer, 848
- getWidgetLook
  - CEGUI::WidgetLookManager, 1219
- getWidth
  - CEGUI::Image, 432
  - CEGUI::Renderer, 796
  - CEGUI::Texture, 1092
  - CEGUI::Window, 1281
- getWindow
  - CEGUI::ScriptWindowHelper, 831
  - CEGUI::WindowManager, 1313
- getWindowContainingMouse
  - CEGUI::System, 1050
- getWindowRenderer
  - CEGUI::Window, 1283
- getWindowRendererName
  - CEGUI::Window, 1283
- getWordStartIdx
  - CEGUI::TextUtils, 1096
- getXPosition
  - CEGUI::Window, 1280
- getXScale
  - CEGUI::Texture, 1093
- getYPosition
  - CEGUI::Window, 1280
- getYScale
  - CEGUI::Texture, 1093
- getZLayer
  - CEGUI::Renderer, 797
- HA\_CENTRE
  - CEGUI, 47
- HA\_LEFT
  - CEGUI, 47



- HA\_RIGHT
  - CEGUI, [47](#)
- handleThumbMoved
  - CEGUI::Scrollbar, [857](#)
  - CEGUI::Slider, [918](#)
- handleUpdatedItemData
  - CEGUI::ItemListBase, [497](#)
  - CEGUI::Listbox, [532](#)
  - CEGUI::MultiColumnList, [668](#)
  - CEGUI::Tree, [1138](#)
- handleUpdatedListItemData
  - CEGUI::Combobox, [169](#)
- hasCachedImagery
  - CEGUI::RenderCache, [789](#)
- hasInputFocus
  - CEGUI::Combobox, [160](#)
  - CEGUI::Editbox, [259](#)
  - CEGUI::MultiLineEditbox, [684](#)
- Hexadecimal
  - CEGUI::Spinner, [952](#)
- HF\_CENTRE\_ALIGNED
  - CEGUI, [47](#)
- HF\_LEFT\_ALIGNED
  - CEGUI, [47](#)
- HF\_RIGHT\_ALIGNED
  - CEGUI, [47](#)
- HF\_STRETCHED
  - CEGUI, [47](#)
- HF\_TILED
  - CEGUI, [47](#)
- hide
  - CEGUI::MouseCursor, [629](#)
  - CEGUI::Window, [1264](#)
- hideDropList
  - CEGUI::Combobox, [164](#)
- HorizontalAlignment
  - CEGUI, [47](#)
- HorizontalFormatting
  - CEGUI, [47](#)
- HorizontalTextFormatting
  - CEGUI, [47](#)
- HTF\_CENTRE\_ALIGNED
  - CEGUI, [48](#)
- HTF\_JUSTIFIED
  - CEGUI, [48](#)
- HTF\_LEFT\_ALIGNED
  - CEGUI, [48](#)
- HTF\_RIGHT\_ALIGNED
  - CEGUI, [48](#)
- HTF\_WORDWRAP\_CENTRE\_ALIGNED
  - CEGUI, [48](#)
- HTF\_WORDWRAP\_JUSTIFIED
  - CEGUI, [48](#)
- HTF\_WORDWRAP\_LEFT\_ALIGNED
  - CEGUI, [48](#)
- HTF\_WORDWRAP\_RIGHT\_ALIGNED
  - CEGUI, [48](#)
- Image
  - CEGUI::Image, [432](#)
- ImageDim
  - CEGUI::ImageDim, [442](#)
- ImagerySection
  - CEGUI::ImagerySection, [449](#)
- Imageset\_xmlHandler
  - CEGUI::Imageset\_xmlHandler, [467](#)
- Inactive
  - CEGUI::Tooltip, [1117](#)
- Informative
  - CEGUI, [48](#)
- inheritsAlpha
  - CEGUI::Window, [1251](#)
- inheritsTooltipText
  - CEGUI::Window, [1258](#)
- initColourRectForOverride
  - CEGUI::SectionSpecification, [877](#)
- initColoursRect
  - CEGUI::FalagardComponentBase, [303](#)
- initialise
  - CEGUI::Tree, [1134](#)
  - CEGUI::XMLParser, [1334](#)
- initialiseComponents
  - CEGUI::Combobox, [164](#)
  - CEGUI::ComboDropList, [173](#)
  - CEGUI::FrameWindow, [374](#)
  - CEGUI::ItemListBase, [495](#)
  - CEGUI::Listbox, [529](#)
  - CEGUI::MultiColumnList, [660](#)
  - CEGUI::MultiLineEditbox, [687](#)
  - CEGUI::ScrollablePane, [842](#)
  - CEGUI::Scrollbar, [854](#)
  - CEGUI::ScrolledItemBase, [870](#)
  - CEGUI::Slider, [916](#)
  - CEGUI::Spinner, [952](#)
  - CEGUI::TabControl, [1068](#)
  - CEGUI::Window, [1262](#)
- initialiseDragging
  - CEGUI::DragContainer, [233](#)
- initialiseImpl
  - CEGUI::XMLParser, [1335](#)
- initialiseWidget
  - CEGUI::WidgetLookFeel, [1212](#)
- initMasterColourRect
  - CEGUI::ImagerySection, [453](#)
- injectChar
  - CEGUI::System, [1056](#)
- injectKeyDown
  - CEGUI::System, [1056](#)

- injectKeyUp
  - CEGUI::System, 1056
- injectMouseButtonDown
  - CEGUI::System, 1055
- injectMouseButtonUp
  - CEGUI::System, 1055
- injectMouseLeaves
  - CEGUI::System, 1055
- injectMouseMove
  - CEGUI::System, 1055
- injectMousePosition
  - CEGUI::System, 1057
- injectMouseWheelChange
  - CEGUI::System, 1056
- injectTimePulse
  - CEGUI::System, 1057
- Insane
  - CEGUI, 48
- insert
  - CEGUI::String, 1000–1005
- insertColumn
  - CEGUI::ListHeader, 565
  - CEGUI::MultiColumnList, 660
- insertItem
  - CEGUI::Combobox, 167
  - CEGUI::ItemListBase, 496
  - CEGUI::Listbox, 529
  - CEGUI::Tree, 1135
- insertRow
  - CEGUI::MultiColumnList, 663
- Integer
  - CEGUI::Spinner, 952
- InvalidRequestException
  - CEGUI::InvalidRequestException, 478
- isActive
  - CEGUI::Window, 1247
- isAlwaysOnTop
  - CEGUI::Window, 1246
- isAncestor
  - CEGUI::Window, 1250, 1251
- isAreaFetchedFromProperty
  - CEGUI::ComponentArea, 178
- isArmed
  - CEGUI::ComboDropList, 174
- isAutoArmEnabled
  - CEGUI::ComboDropList, 174
- isAutoDeleted
  - CEGUI::ListboxItem, 541
  - CEGUI::TreeItem, 1150
- isAutoResizeEnabled
  - CEGUI::ItemListBase, 495
- isAutoScaled
  - CEGUI::Imageset, 462
- isBeingDragged
  - CEGUI::DragContainer, 230
- isBottomSizingLocation
  - CEGUI::FrameWindow, 383
- isCapturedByAncestor
  - CEGUI::Window, 1254
- isCapturedByChild
  - CEGUI::Window, 1254
- isCapturedByThis
  - CEGUI::Window, 1254
- isChild
  - CEGUI::Window, 1247, 1248
- isChildRecursive
  - CEGUI::Window, 1248
- isClickable
  - CEGUI::ListHeaderSegment, 577
- isClippedByParent
  - CEGUI::Window, 1247
- isClippedToDisplay
  - CEGUI::StateImagery, 961
- isCloseButtonEnabled
  - CEGUI::FrameWindow, 375
- isCodepointAvailable
  - CEGUI::Font, 317
- isColumnDraggingEnabled
  - CEGUI::ListHeader, 562
- isColumnSizingEnabled
  - CEGUI::ListHeader, 562
- isContentPaneAutoSized
  - CEGUI::ScrollablePane, 838
  - CEGUI::ScrolledContainer, 864
- isDeadPoolEmpty
  - CEGUI::WindowManager, 1315
- isDefault
  - CEGUI::Property, 746
  - CEGUI::WindowProperties::Disabled, 215
  - CEGUI::WindowProperties::Font, 310
  - CEGUI::WindowProperties::MouseCursorImage, 633
  - CEGUI::WindowProperties::Visible, 1201
- isDestroyedByParent
  - CEGUI::Window, 1246
- isDisabled
  - CEGUI::ListboxItem, 541
  - CEGUI::TreeItem, 1150
  - CEGUI::Window, 1246
- isDragDropTarget
  - CEGUI::Window, 1262
- isDraggingEnabled
  - CEGUI::DragContainer, 230
  - CEGUI::Titlebar, 1107
- isDraggingThresholdExceeded
  - CEGUI::DragContainer, 233
- isDragMoveThresholdExceeded
  - CEGUI::ListHeaderSegment, 578



- isDragMovingEnabled
  - CEGUI::FrameWindow, 377
  - CEGUI::ListHeaderSegment, 576
- isDropDownListVisible
  - CEGUI::Combobox, 159
- isEventPresent
  - CEGUI::EventSet, 281
- isFactoryPresent
  - CEGUI::WindowFactoryManager, 1304
- isFalagardMappedType
  - CEGUI::WindowFactoryManager, 1306
- isFontFetchedFromProperty
  - CEGUI::TextComponent, 1088
- isFontPresent
  - CEGUI::FontManager, 334
- isFrameEnabled
  - CEGUI::FrameWindow, 374
- isHit
  - CEGUI::Combobox, 159
  - CEGUI::FrameWindow, 380
  - CEGUI::Window, 1254
- isHorzFree
  - CEGUI::Thumb, 1101
- isHorzScrollbarAlwaysShown
  - CEGUI::Combobox, 164
  - CEGUI::Listbox, 529
  - CEGUI::MultiColumnList, 658
  - CEGUI::ScrollablePane, 837
  - CEGUI::Tree, 1134
- isHorzScrollbarNeeded
  - CEGUI::ScrollablePane, 843
- isHotTracked
  - CEGUI::Thumb, 1100
- isHovering
  - CEGUI::ButtonBase, 110
  - CEGUI::MenuItem, 615
- isImageDefined
  - CEGUI::Imageset, 458
- isImageFetchedFromProperty
  - CEGUI::ImageryComponent, 446
- isImagesetPresent
  - CEGUI::ImagesetManager, 471
- isItemInList
  - CEGUI::ItemListBase, 495
- isItemSelected
  - CEGUI::Combobox, 163
  - CEGUI::Listbox, 528
  - CEGUI::MultiColumnList, 656
- isLeftSizingLocation
  - CEGUI::FrameWindow, 382
- isListboxItemInColumn
  - CEGUI::MultiColumnList, 653
- isListboxItemInList
  - CEGUI::Combobox, 163
  - CEGUI::Listbox, 528
  - CEGUI::MultiColumnList, 654
- isListboxItemInRow
  - CEGUI::MultiColumnList, 654
- isMonochromatic
  - CEGUI::ColourRect, 140
- isMouseAutoRepeatEnabled
  - CEGUI::Window, 1256
- isMousePassThroughEnabled
  - CEGUI::Window, 1261
- isMultiplePopupsAllowed
  - CEGUI::MenuBase, 611
- isMultiselectEnabled
  - CEGUI::Listbox, 527
  - CEGUI::Tree, 1133
- isMuted
  - CEGUI::EventSet, 283
- isNamedAreaDefined
  - CEGUI::WidgetLookFeel, 1214
- isPointInRect
  - CEGUI::Rect, 783
- isPropertyDefault
  - CEGUI::PropertySet, 766
- isPropertyPresent
  - CEGUI::PropertySet, 765
- isPushed
  - CEGUI::ButtonBase, 110
  - CEGUI::MenuItem, 615
- isQueueingEnabled
  - CEGUI::Renderer, 795
- isReadOnly
  - CEGUI::Combobox, 160
  - CEGUI::Editbox, 259
  - CEGUI::MultiLineEditbox, 685
- isRedrawRequested
  - CEGUI::System, 1047
- isRightSizingLocation
  - CEGUI::FrameWindow, 382
- isRiseOnClickEnabled
  - CEGUI::Window, 1258
- isRolledup
  - CEGUI::FrameWindow, 375
- isRollupEnabled
  - CEGUI::FrameWindow, 375
- isSchemePresent
  - CEGUI::SchemeManager, 820
- isSelected
  - CEGUI::Checkbox, 120
  - CEGUI::ListboxItem, 541
  - CEGUI::RadioButton, 771
  - CEGUI::TreeItem, 1150
- isSizingEnabled
  - CEGUI::FrameWindow, 374
  - CEGUI::ListHeaderSegment, 576

- isSortEnabled
  - CEGUI::Combobox, 163
  - CEGUI::Listbox, 527
  - CEGUI::Tree, 1133
- isSortingEnabled
  - CEGUI::ListHeader, 562
- isStateImageryPresent
  - CEGUI::WidgetLookFeel, 1213
- isTabContentsSelected
  - CEGUI::TabControl, 1068
- isTextFetchedFromProperty
  - CEGUI::TextComponent, 1087
- isTextMasked
  - CEGUI::Editbox, 259
- isTextValid
  - CEGUI::Combobox, 160
  - CEGUI::Editbox, 260
- isTitleBarEnabled
  - CEGUI::FrameWindow, 374
- isTopOfZOrder
  - CEGUI::Window, 1297
- isTopSizingLocation
  - CEGUI::FrameWindow, 383
- isTreeItemInList
  - CEGUI::Tree, 1134
- isUserColumnDraggingEnabled
  - CEGUI::MultiColumnList, 650
- isUserColumnSizingEnabled
  - CEGUI::MultiColumnList, 650
- isUserSortControlEnabled
  - CEGUI::MultiColumnList, 650
- isUserStringDefined
  - CEGUI::Window, 1260
- isUsingDefaultTooltip
  - CEGUI::Window, 1257
- isUsingOverrideColours
  - CEGUI::SectionSpecification, 876
- isVertFree
  - CEGUI::Thumb, 1101
- isVertScrollbarAlwaysShown
  - CEGUI::Combobox, 164
  - CEGUI::Listbox, 528
  - CEGUI::MultiColumnList, 658
  - CEGUI::MultiLineEditbox, 686
  - CEGUI::ScrollablePane, 837
  - CEGUI::Tree, 1134
- isVertScrollbarNeeded
  - CEGUI::ScrollablePane, 842
- isVisible
  - CEGUI::MouseCursor, 629
  - CEGUI::Window, 1246
- isWidgetLookAvailable
  - CEGUI::WidgetLookManager, 1219
- isWindowPresent
  - CEGUI::WindowManager, 1314
- isWordWrapped
  - CEGUI::MultiLineEditbox, 686
- isZOrderingEnabled
  - CEGUI::Window, 1256
- Justified
  - CEGUI, 49
- LayerSpecification
  - CEGUI::LayerSpecification, 518
- layoutChildWidgets
  - CEGUI::WidgetLookFeel, 1214
- layoutItemWidgets
  - CEGUI::ItemListBase, 499
  - CEGUI::Menubar, 608
  - CEGUI::PopupMenu, 731
- LeftAligned
  - CEGUI, 49
- LeftMouse
  - CEGUI, 49
- length
  - CEGUI::String, 982
- load
  - CEGUI::ImageCodec, 440
  - CEGUI::ImageSet, 464
- loadFromFile
  - CEGUI::Texture, 1093
- loadFromMemory
  - CEGUI::Texture, 1094
- loadRawDataContainer
  - CEGUI::ResourceProvider, 802
- loadResources
  - CEGUI::Scheme, 815
- loadScheme
  - CEGUI::SchemeManager, 819
- loadWindowLayout
  - CEGUI::WindowManager, 1314
- logEvent
  - CEGUI::DefaultLogger, 208
  - CEGUI::Logger, 585
- LoggingLevel
  - CEGUI, 48
- makeTabVisible\_impl
  - CEGUI::TabControl, 1070
- max\_size
  - CEGUI::String, 982
- MemoryException
  - CEGUI::MemoryException, 605
- MiddleMouse
  - CEGUI, 49
- modulateAlpha
  - CEGUI::ColourRect, 141

- MouseButton
  - CEGUI, 48
- MouseCursorImage
  - CEGUI, 48
- moveBottomEdge
  - CEGUI::FrameWindow, 382
- moveColumn
  - CEGUI::ListHeader, 566, 567
  - CEGUI::MultiColumnList, 661
- moveColumn\_impl
  - CEGUI::MultiColumnList, 671
- moveColumnWithID
  - CEGUI::MultiColumnList, 662
- moveLeftEdge
  - CEGUI::FrameWindow, 381
- moveRightEdge
  - CEGUI::FrameWindow, 381
- moveSegment
  - CEGUI::ListHeader, 567
- moveToBack
  - CEGUI::Window, 1268
- moveToFront
  - CEGUI::Window, 1267
- moveToFront\_impl
  - CEGUI::Window, 1296
- moveTopEdge
  - CEGUI::FrameWindow, 381
- NoButton
  - CEGUI, 48
- None
  - CEGUI::ListHeaderSegment, 576
- notifyScreenResolution
  - CEGUI::Font, 321
  - CEGUI::FontManager, 334
  - CEGUI::Imageset, 463
  - CEGUI::ImagesetManager, 471
- notifyWindowDestroyed
  - CEGUI::System, 1052
- NullPointerException
  - CEGUI::NullPointerException, 715
- ObjectInUseException
  - CEGUI::ObjectInUseException, 719
- Octal
  - CEGUI::Spinner, 952
- offset
  - CEGUI::Rect, 783
- offsetPixelPosition
  - CEGUI::FrameWindow, 377
- offsetPosition
  - CEGUI::MouseCursor, 628
- onActivated
  - CEGUI::Combobox, 170
  - CEGUI::ComboDropList, 176
  - CEGUI::FrameWindow, 384
  - CEGUI::Spinner, 956
  - CEGUI::Window, 1288
- onAlphaChanged
  - CEGUI::DragContainer, 235
  - CEGUI::PopupMenu, 732
  - CEGUI::Window, 1285
- onAlwaysOnTopChanged
  - CEGUI::Window, 1287
- onAutoSizeSettingChanged
  - CEGUI::ScrollablePane, 844
  - CEGUI::ScrolledContainer, 866
- onCaptureGained
  - CEGUI::Window, 1287
- onCaptureLost
  - CEGUI::ButtonBase, 111
  - CEGUI::ComboDropList, 175
  - CEGUI::DragContainer, 234
  - CEGUI::Editbox, 265
  - CEGUI::FrameWindow, 384
  - CEGUI::ListHeaderSegment, 580
  - CEGUI::MenuItem, 617
  - CEGUI::MultiLineEditbox, 691
  - CEGUI::Thumb, 1103
  - CEGUI::Titlebar, 1108
  - CEGUI::Window, 1287
- onCharacter
  - CEGUI::Editbox, 265
  - CEGUI::MultiLineEditbox, 691
  - CEGUI::Window, 1292
- onChildAdded
  - CEGUI::ScrolledContainer, 866
  - CEGUI::Window, 1289
- onChildRemoved
  - CEGUI::ScrolledContainer, 867
  - CEGUI::Window, 1289
- onClippingChanged
  - CEGUI::DragContainer, 235
  - CEGUI::Window, 1286
- onContentChanged
  - CEGUI::ScrolledContainer, 866
- onContentPaneChanged
  - CEGUI::ScrollablePane, 844
- onContentPaneScrolled
  - CEGUI::ScrollablePane, 845
- onDeactivated
  - CEGUI::FrameWindow, 385
  - CEGUI::Window, 1288
- onDestructionStarted
  - CEGUI::PopupMenu, 732
  - CEGUI::Window, 1288
- onDisabled
  - CEGUI::Window, 1286

- onDisplayTimeChanged
  - CEGUI::Tooltip, [1121](#)
- onDragAlphaChanged
  - CEGUI::DragContainer, [236](#)
- onDragDropItemDropped
  - CEGUI::Window, [1293](#)
- onDragDropItemEnters
  - CEGUI::Window, [1292](#)
- onDragDropItemLeaves
  - CEGUI::Window, [1292](#)
- onDragDropTargetChanged
  - CEGUI::DragContainer, [237](#)
- onDragEnabledChanged
  - CEGUI::DragContainer, [236](#)
- onDragEnded
  - CEGUI::DragContainer, [236](#)
- onDraggingModeChanged
  - CEGUI::Titlebar, [1108](#)
- onDragMouseCursorChanged
  - CEGUI::DragContainer, [237](#)
- onDragPositionChanged
  - CEGUI::DragContainer, [236](#)
- onDragStarted
  - CEGUI::DragContainer, [235](#)
- onDragThresholdChanged
  - CEGUI::DragContainer, [237](#)
- onEnabled
  - CEGUI::Window, [1286](#)
- onFadeTimeChanged
  - CEGUI::Tooltip, [1121](#)
- onFontChanged
  - CEGUI::Combobox, [169](#)
  - CEGUI::MultiColumnList, [672](#)
  - CEGUI::Spinner, [956](#)
  - CEGUI::TabControl, [1071](#)
  - CEGUI::Titlebar, [1108](#)
  - CEGUI::Window, [1285](#)
- onHidden
  - CEGUI::PopupMenu, [733](#)
  - CEGUI::Window, [1285](#)
- onHorizontalAlignmentChanged
  - CEGUI::Window, [1293](#)
- onHorzScrollbarModeChanged
  - CEGUI::ScrollablePane, [844](#)
- onHoverTimeChanged
  - CEGUI::Tooltip, [1120](#)
- onIDChanged
  - CEGUI::Window, [1285](#)
- onInheritsAlphaChanged
  - CEGUI::Window, [1286](#)
- onKeyDown
  - CEGUI::Editbox, [266](#)
  - CEGUI::ItemListbox, [508](#)
  - CEGUI::MultiLineEditbox, [692](#)
  - CEGUI::Window, [1291](#)
- onKeyUp
  - CEGUI::Window, [1292](#)
- onMaximumValueChanged
  - CEGUI::Spinner, [957](#)
- onMinimumValueChanged
  - CEGUI::Spinner, [957](#)
- onMouseButtonDown
  - CEGUI::ButtonBase, [110](#)
  - CEGUI::ComboDropList, [175](#)
  - CEGUI::DragContainer, [234](#)
  - CEGUI::Editbox, [264](#)
  - CEGUI::FrameWindow, [384](#)
  - CEGUI::Listbox, [535](#)
  - CEGUI::ListHeaderSegment, [579](#)
  - CEGUI::MenuItem, [617](#)
  - CEGUI::MultiColumnList, [672](#)
  - CEGUI::MultiLineEditbox, [690](#)
  - CEGUI::PopupMenu, [733](#)
  - CEGUI::Scrollbar, [858](#)
  - CEGUI::Slider, [919](#)
  - CEGUI::TabButton, [1060](#)
  - CEGUI::Thumb, [1103](#)
  - CEGUI::Titlebar, [1107](#)
  - CEGUI::Tree, [1141](#)
  - CEGUI::Window, [1290](#)
- onMouseButtonUp
  - CEGUI::ButtonBase, [111](#)
  - CEGUI::Checkbox, [120](#)
  - CEGUI::ComboDropList, [175](#)
  - CEGUI::DragContainer, [234](#)
  - CEGUI::Editbox, [264](#)
  - CEGUI::FrameWindow, [384](#)
  - CEGUI::ListHeaderSegment, [579](#)
  - CEGUI::MenuItem, [617](#)
  - CEGUI::MultiLineEditbox, [690](#)
  - CEGUI::PopupMenu, [733](#)
  - CEGUI::PushButton, [768](#)
  - CEGUI::RadioButton, [773](#)
  - CEGUI::TabButton, [1060](#)
  - CEGUI::Titlebar, [1108](#)
  - CEGUI::Window, [1290](#)
- onMouseClicked
  - CEGUI::ItemEntry, [484](#)
  - CEGUI::Window, [1291](#)
- onMouseDoubleClicked
  - CEGUI::Editbox, [265](#)
  - CEGUI::ListHeaderSegment, [580](#)
  - CEGUI::MultiLineEditbox, [690](#)
  - CEGUI::Titlebar, [1108](#)
  - CEGUI::Window, [1291](#)
- onMouseEnters
  - CEGUI::Tooltip, [1122](#)
  - CEGUI::Window, [1289](#)

- onMouseLeaves
  - CEGUI::ButtonBase, 111
  - CEGUI::ListHeaderSegment, 580
  - CEGUI::MenuItem, 617
  - CEGUI::Window, 1289
- onMouseMove
  - CEGUI::ButtonBase, 110
  - CEGUI::ComboDropList, 175
  - CEGUI::DragContainer, 234
  - CEGUI::Editbox, 265
  - CEGUI::FrameWindow, 383
  - CEGUI::Listbox, 535
  - CEGUI::ListHeaderSegment, 579
  - CEGUI::MenuItem, 617
  - CEGUI::MultiLineEditbox, 691
  - CEGUI::TabButton, 1061
  - CEGUI::Thumb, 1103
  - CEGUI::Titlebar, 1107
  - CEGUI::Tree, 1141
  - CEGUI::Window, 1290
- onMouseTripleClicked
  - CEGUI::Editbox, 265
  - CEGUI::MultiLineEditbox, 691
  - CEGUI::Window, 1291
- onMouseWheel
  - CEGUI::Listbox, 535
  - CEGUI::MultiColumnList, 672
  - CEGUI::MultiLineEditbox, 692
  - CEGUI::ScrollablePane, 845
  - CEGUI::Scrollbar, 858
  - CEGUI::ScrolledItemListBase, 870
  - CEGUI::Slider, 919
  - CEGUI::TabButton, 1060
  - CEGUI::Tree, 1141
  - CEGUI::Window, 1290
- onMoved
  - CEGUI::DragContainer, 235
  - CEGUI::Window, 1284
- onParentDestroyChanged
  - CEGUI::Window, 1286
- onParentSized
  - CEGUI::ScrolledContainer, 867
  - CEGUI::Window, 1289
- onRenderingEnded
  - CEGUI::Window, 1287
- onRenderingStarted
  - CEGUI::Window, 1287
- onShown
  - CEGUI::PopupMenu, 733
  - CEGUI::Window, 1285
- onSized
  - CEGUI::Listbox, 535
  - CEGUI::MultiColumnList, 672
  - CEGUI::MultiLineEditbox, 692
  - CEGUI::ScrollablePane, 845
  - CEGUI::Tree, 1141
  - CEGUI::Window, 1284
- onStepChanged
  - CEGUI::Spinner, 957
- onTextChanged
  - CEGUI::Combobox, 170
  - CEGUI::Editbox, 266
  - CEGUI::FrameWindow, 384
  - CEGUI::MenuItem, 618
  - CEGUI::MultiLineEditbox, 692
  - CEGUI::Spinner, 956
  - CEGUI::Tooltip, 1122
  - CEGUI::Window, 1284
- onTextInputModeChanged
  - CEGUI::Spinner, 958
- onTextInvalidatedEvent
  - CEGUI::Editbox, 264
- onTooltipActive
  - CEGUI::Tooltip, 1121
- onTooltipInactive
  - CEGUI::Tooltip, 1121
- onValueChanged
  - CEGUI::Spinner, 957
- onVerticalAlignmentChanged
  - CEGUI::Window, 1293
- onVertScrollbarModeChanged
  - CEGUI::ScrollablePane, 844
- onWindowRendererAttached
  - CEGUI::Window, 1293
- onWindowRendererDetached
  - CEGUI::Window, 1294
- onZChanged
  - CEGUI::Window, 1288
- openPopupMenu
  - CEGUI::MenuItem, 616
  - CEGUI::PopupMenu, 731
- openTag
  - CEGUI::XMLSerializer, 1337
- operator bool
  - CEGUI::XMLSerializer, 1338
- operator!
  - CEGUI::XMLSerializer, 1338
- operator!=
  - CEGUI::BoundSlot, 108
- operator()
  - CEGUI::Event, 276
- operator+
  - CEGUI, 51–53
- operator++
  - CEGUI::ConstBaseIterator, 183
- operator+=
  - CEGUI::String, 995–999
- operator–

- CEGUI::ConstBaseIterator, 184
- operator=
  - CEGUI::String, 991–994
- operator==
  - CEGUI::BoundSlot, 107
- OrientationFlags
  - CEGUI, 48
- parseLookNFeelSpecification
  - CEGUI::WidgetLookManager, 1218
- parseXMLFile
  - CEGUI::XMLParser, 1334
- performChildWindowLayout
  - CEGUI::ItemListBase, 497
  - CEGUI::TabControl, 1071
  - CEGUI::Window, 1275
- PF\_RGB
  - CEGUI::Texture, 1092
- PF\_RGBA
  - CEGUI::Texture, 1092
- PixelFormat
  - CEGUI::Texture, 1092
- PixmapFont
  - CEGUI::PixmapFont, 727
- populateRenderCache
  - CEGUI::Tree, 1141
  - CEGUI::Window, 1294
- positionSelf
  - CEGUI::Tooltip, 1119
- Property
  - CEGUI::Property, 742
- PropertyCallback
  - CEGUI::WindowManager, 1312
- PropertyDim
  - CEGUI::PropertyDim, 755
- PropertyInitialiser
  - CEGUI::PropertyInitialiser, 757
- PropertyList
  - CEGUI::WidgetLookFeel, 1210
- push\_back
  - CEGUI::String, 998
- QuadSplitMode
  - CEGUI, 49
- rasterize
  - CEGUI::Font, 317
- rbegin
  - CEGUI::String, 1033, 1034
- registerAllFactories
  - CEGUI::FactoryModule, 291
- registerFactory
  - CEGUI::FactoryModule, 291
- registerProperty
  - CEGUI::WindowRenderer, 1321
- releaseInput
  - CEGUI::Window, 1268
- remove
  - CEGUI::XMLAttributes, 1329
- removeAllEvents
  - CEGUI::EventSet, 281
- removeAllFactories
  - CEGUI::WindowFactoryManager, 1303
- removeChildWindow
  - CEGUI::Window, 1267
- removeColumn
  - CEGUI::ListHeader, 566
  - CEGUI::MultiColumnList, 661
- removeColumnWithID
  - CEGUI::MultiColumnList, 661
- removeEvent
  - CEGUI::EventSet, 281
- removeFactory
  - CEGUI::WindowFactoryManager, 1303
- removeFalagardWindowMapping
  - CEGUI::WindowFactoryManager, 1306
- removeItem
  - CEGUI::Combobox, 167
  - CEGUI::ItemListBase, 496
  - CEGUI::Listbox, 530
  - CEGUI::Tree, 1135
- removeProperty
  - CEGUI::PropertySet, 764
- removeRow
  - CEGUI::MultiColumnList, 664
- removeSegment
  - CEGUI::ListHeader, 566
- removeTab
  - CEGUI::TabControl, 1069
- removeWindowFromDrawList
  - CEGUI::Window, 1297
- removeWindowTypeAlias
  - CEGUI::WindowFactoryManager, 1305
- rename
  - CEGUI::Window, 1262
- renameChildren
  - CEGUI::WidgetLookFeel, 1215
- renameWindow
  - CEGUI::WindowManager, 1316
- rend
  - CEGUI::String, 1034
- render
  - CEGUI::FalagardComponentBase, 301
  - CEGUI::ImagerySection, 450
  - CEGUI::LayerSpecification, 519
  - CEGUI::RenderCache, 789
  - CEGUI::SectionSpecification, 875
  - CEGUI::StateImagery, 960



- CEGUI::Window, 1282
- CEGUI::WindowRenderer, 1321
- RendererException
  - CEGUI::RendererException, 799
- renderGUI
  - CEGUI::System, 1047
- replace
  - CEGUI::String, 1007–1015
- requestRedraw
  - CEGUI::Window, 1269
- reserve
  - CEGUI::String, 983
- resetList
  - CEGUI::Combobox, 166
  - CEGUI::ItemListBase, 496
  - CEGUI::Listbox, 529
  - CEGUI::MultiColumnList, 660
  - CEGUI::Tree, 1135
- resetList\_impl
  - CEGUI::ItemListBase, 499
  - CEGUI::Listbox, 534
  - CEGUI::MultiColumnList, 671
  - CEGUI::Tree, 1140
- resetTimer
  - CEGUI::Tooltip, 1118
- resetZValue
  - CEGUI::Renderer, 797
- resize
  - CEGUI::String, 1006, 1007
- resourcesLoaded
  - CEGUI::Scheme, 815
- restoresOldCapture
  - CEGUI::Window, 1256
- rfind
  - CEGUI::String, 1015–1020
- RightAligned
  - CEGUI, 49
- RightMouse
  - CEGUI, 49
- RotateRightAngle
  - CEGUI, 49
- Scheme\_xmlHandler
  - CEGUI::Scheme\_xmlHandler, 818
- screenToWindow
  - CEGUI::CoordConverter, 197, 198
- screenToWindowX
  - CEGUI::CoordConverter, 196
- screenToWindowY
  - CEGUI::CoordConverter, 196, 197
- ScriptException
  - CEGUI::ScriptException, 822
- SectionSpecification
  - CEGUI::SectionSpecification, 874
- selectRange
  - CEGUI::ItemListbox, 507
- selectTab\_impl
  - CEGUI::TabControl, 1069
- set
  - CEGUI::CheckboxProperties::Selected, 886
  - CEGUI::ComboboxProperties::CaratIndex, 116
  - CEGUI::ComboboxProperties::EditSelectionLength, 271
  - CEGUI::ComboboxProperties::EditSelectionStart, 273
  - CEGUI::ComboboxProperties::ForceHorzScrollbar, 341
  - CEGUI::ComboboxProperties::ForceVertScrollbar, 355
  - CEGUI::ComboboxProperties::MaxEditTextLength, 594
  - CEGUI::ComboboxProperties::ReadOnly, 779
  - CEGUI::ComboboxProperties::SingleClickMode, 903
  - CEGUI::ComboboxProperties::SortList, 941
  - CEGUI::ComboboxProperties::ValidationString, 1183
  - CEGUI::DragContainerProperties::DragAlpha, 225
  - CEGUI::DragContainerProperties::DragCursorImage, 240
  - CEGUI::DragContainerProperties::DraggingEnabled, 247
  - CEGUI::DragContainerProperties::DragThreshold, 251
  - CEGUI::EditboxProperties::ActiveSelectionColour, 87
  - CEGUI::EditboxProperties::CaratIndex, 118
  - CEGUI::EditboxProperties::InactiveSelectionColour, 473
  - CEGUI::EditboxProperties::MaskCodepoint, 590
  - CEGUI::EditboxProperties::MaskText, 592
  - CEGUI::EditboxProperties::MaxTextLength, 602
  - CEGUI::EditboxProperties::NormalTextColour, 712
  - CEGUI::EditboxProperties::ReadOnly, 777
  - CEGUI::EditboxProperties::SelectedTextColour, 888
  - CEGUI::EditboxProperties::SelectionLength, 894
  - CEGUI::EditboxProperties::SelectionStart, 898
  - CEGUI::EditboxProperties::ValidationString, 1185

- CEGUI::FrameWindowProperties::CloseButtonEnabled, 136
- CEGUI::FrameWindowProperties::DragMovingEnabled, 249
- CEGUI::FrameWindowProperties::EWSizingCursorImage, 286
- CEGUI::FrameWindowProperties::FrameEnabled, 367
- CEGUI::FrameWindowProperties::NESWSizingCursorImage, 706
- CEGUI::FrameWindowProperties::NSSizingCursorImage, 714
- CEGUI::FrameWindowProperties::NWSEizingCursorImage, 718
- CEGUI::FrameWindowProperties::RollUpEnabled, 809
- CEGUI::FrameWindowProperties::RollUpState, 811
- CEGUI::FrameWindowProperties::SizingBorderThickness, 908
- CEGUI::FrameWindowProperties::SizingEnabled, 912
- CEGUI::FrameWindowProperties::TitlebarEnabled, 1111
- CEGUI::ItemEntryProperties::Selectable, 880
- CEGUI::ItemEntryProperties::Selected, 884
- CEGUI::ItemListBaseProperties::AutoResizeEnabled, 101
- CEGUI::ItemListBaseProperties::SortEnabled, 939
- CEGUI::ItemListBaseProperties::SortMode, 943
- CEGUI::ItemListboxProperties::MultiSelect, 700
- CEGUI::ListboxProperties::ForceHorzScrollbar, 345
- CEGUI::ListboxProperties::ForceVertScrollbar, 357
- CEGUI::ListboxProperties::ItemTooltips, 514
- CEGUI::ListboxProperties::MultiSelect, 702
- CEGUI::ListboxProperties::Sort, 927
- CEGUI::ListHeaderProperties::ColumnsMovable, 145
- CEGUI::ListHeaderProperties::ColumnsSizable, 149
- CEGUI::ListHeaderProperties::SortColumnID, 931
- CEGUI::ListHeaderProperties::SortDirection, 935
- CEGUI::ListHeaderProperties::SortSettingEnabled, 947
- CEGUI::ListHeaderSegmentProperties::Clickable, 125
- CEGUI::ListHeaderSegmentProperties::Dragable, 223
- CEGUI::ListHeaderSegmentProperties::MovingCursorImage, 639
- CEGUI::ListHeaderSegmentProperties::Sizable, 905
- CEGUI::ListHeaderSegmentProperties::SizingCursorImage, 910
- CEGUI::ListHeaderSegmentProperties::SortDirection, 933
- CEGUI::MenuBaseProperties::AllowMultiplePopups, 89
- CEGUI::MenuBaseProperties::ItemSpacing, 510
- CEGUI::MultiColumnListProperties::ColumnHeader, 143
- CEGUI::MultiColumnListProperties::ColumnsMovable, 147
- CEGUI::MultiColumnListProperties::ColumnsSizable, 151
- CEGUI::MultiColumnListProperties::ForceHorzScrollbar, 347
- CEGUI::MultiColumnListProperties::ForceVertScrollbar, 361
- CEGUI::MultiColumnListProperties::NominatedSelectionColumnID, 708
- CEGUI::MultiColumnListProperties::NominatedSelectionRow, 710
- CEGUI::MultiColumnListProperties::RowCount, 813
- CEGUI::MultiColumnListProperties::SelectionMode, 896
- CEGUI::MultiColumnListProperties::SortColumnID, 929
- CEGUI::MultiColumnListProperties::SortDirection, 937
- CEGUI::MultiColumnListProperties::SortSettingEnabled, 945
- CEGUI::MultiLineEditboxProperties::CaratIndex, 114
- CEGUI::MultiLineEditboxProperties::ForceVertScrollbar, 349
- CEGUI::MultiLineEditboxProperties::MaxTextLength, 600
- CEGUI::MultiLineEditboxProperties::ReadOnly, 781
- CEGUI::MultiLineEditboxProperties::SelectionBrushImage, 890
- CEGUI::MultiLineEditboxProperties::SelectionLength, 892
- CEGUI::MultiLineEditboxProperties::SelectionStart, 900
- CEGUI::MultiLineEditboxProperties::WordWrap, 1327



- CEGUI::PopupMenuProperties::FadeInTime, [293](#)
- CEGUI::PopupMenuProperties::FadeOutTime, [295](#)
- CEGUI::ProgressBarProperties::CurrentProgress, [200](#)
- CEGUI::ProgressBarProperties::StepSize, [963](#)
- CEGUI::Property, [744](#)
- CEGUI::PropertyDefinition, [749](#)
- CEGUI::PropertyDefinitionBase, [752](#)
- CEGUI::PropertyLinkDefinition, [760](#)
- CEGUI::RadioButtonProperties::GroupID, [404](#)
- CEGUI::RadioButtonProperties::Selected, [882](#)
- CEGUI::ScrollablePaneProperties::ContentArea, [186](#)
- CEGUI::ScrollablePaneProperties::ContentPaneAutoSize, [190](#)
- CEGUI::ScrollablePaneProperties::ForceHorzScrollbar, [337](#)
- CEGUI::ScrollablePaneProperties::ForceVertScrollbar, [351](#)
- CEGUI::ScrollablePaneProperties::HorzOverlapSize, [417](#)
- CEGUI::ScrollablePaneProperties::HorzScrollPosition, [421](#)
- CEGUI::ScrollablePaneProperties::HorzStepSize, [423](#)
- CEGUI::ScrollablePaneProperties::VertOverlapSize, [1193](#)
- CEGUI::ScrollablePaneProperties::VertScrollPosition, [1197](#)
- CEGUI::ScrollablePaneProperties::VertStepSize, [1199](#)
- CEGUI::ScrollbarProperties::DocumentSize, [221](#)
- CEGUI::ScrollbarProperties::OverlapSize, [722](#)
- CEGUI::ScrollbarProperties::PageSize, [724](#)
- CEGUI::ScrollbarProperties::ScrollPosition, [872](#)
- CEGUI::ScrollbarProperties::StepSize, [967](#)
- CEGUI::ScrolledContainerProperties::ChildExtentsArea, [123](#)
- CEGUI::ScrolledContainerProperties::ContentArea, [188](#)
- CEGUI::ScrolledContainerProperties::ContentPaneAutoSize, [192](#)
- CEGUI::ScrolledItemListBaseProperties::ForceHorzScrollbar, [339](#)
- CEGUI::ScrolledItemListBaseProperties::ForceVertScrollbar, [353](#)
- CEGUI::SliderProperties::ClickStepSize, [127](#)
- CEGUI::SliderProperties::CurrentValue, [202](#)
- CEGUI::SliderProperties::MaximumValue, [596](#)
- CEGUI::SpinnerProperties::CurrentValue, [204](#)
- CEGUI::SpinnerProperties::MaximumValue, [598](#)
- CEGUI::SpinnerProperties::MinimumValue, [621](#)
- CEGUI::SpinnerProperties::StepSize, [965](#)
- CEGUI::SpinnerProperties::TextInputMode, [1090](#)
- CEGUI::TabControlProperties::TabHeight, [1077](#)
- CEGUI::TabControlProperties::TabPagePosition, [1079](#)
- CEGUI::TabControlProperties::TabTextPadding, [1081](#)
- CEGUI::ThumbProperties::HorzFree, [415](#)
- CEGUI::ThumbProperties::HorzRange, [419](#)
- CEGUI::ThumbProperties::HotTracked, [425](#)
- CEGUI::ThumbProperties::VertFree, [1189](#)
- CEGUI::ThumbProperties::VertRange, [1195](#)
- CEGUI::TitlebarProperties::DraggingEnabled, [245](#)
- CEGUI::TooltipProperties::DisplayTime, [217](#)
- CEGUI::TooltipProperties::FadeTime, [297](#)
- CEGUI::TooltipProperties::HoverTime, [427](#)
- CEGUI::TreeProperties::ForceHorzScrollbar, [343](#)
- CEGUI::TreeProperties::ForceVertScrollbar, [359](#)
- CEGUI::TreeProperties::ItemTooltips, [512](#)
- CEGUI::TreeProperties::MultiSelect, [698](#)
- CEGUI::TreeProperties::Sort, [925](#)
- CEGUI::WindowProperties::Alpha, [91](#)
- CEGUI::WindowProperties::AlwaysOnTop, [95](#)
- CEGUI::WindowProperties::AutoRepeatDelay, [97](#)
- CEGUI::WindowProperties::AutoRepeatRate, [99](#)
- CEGUI::WindowProperties::ClippedByParent, [129](#)
- CEGUI::WindowProperties::CustomTooltipType, [206](#)
- CEGUI::WindowProperties::DestroyedByParent, [210](#)
- CEGUI::WindowProperties::Disabled, [215](#)
- CEGUI::WindowProperties::DistributeCapturedInputs, [219](#)
- CEGUI::WindowProperties::DragDropTarget, [243](#)
- CEGUI::WindowProperties::Font, [310](#)
- CEGUI::WindowProperties::HorizontalAlignment, [413](#)

- CEGUI::WindowProperties::ID, [429](#)
- CEGUI::WindowProperties::InheritsAlpha, [475](#)
- CEGUI::WindowProperties::InheritsTooltipText, [477](#)
- CEGUI::WindowProperties::LookNFeel, [588](#)
- CEGUI::WindowProperties::MouseButtonDownActionDeleted, [623](#)
- CEGUI::WindowProperties::MouseCursorImage, [633](#)
- CEGUI::WindowProperties::MousePassThroughEnabled, [637](#)
- CEGUI::WindowProperties::RestoreOldCapture, [805](#)
- CEGUI::WindowProperties::RiseOnClick, [807](#)
- CEGUI::WindowProperties::Text, [1083](#)
- CEGUI::WindowProperties::Tooltip, [1113](#)
- CEGUI::WindowProperties::UnifiedAreaRect, [1158](#)
- CEGUI::WindowProperties::UnifiedHeight, [1162](#)
- CEGUI::WindowProperties::UnifiedMaxSize, [1164](#)
- CEGUI::WindowProperties::UnifiedMinSize, [1166](#)
- CEGUI::WindowProperties::UnifiedPosition, [1168](#)
- CEGUI::WindowProperties::UnifiedSize, [1170](#)
- CEGUI::WindowProperties::UnifiedWidth, [1172](#)
- CEGUI::WindowProperties::UnifiedXPosition, [1174](#)
- CEGUI::WindowProperties::UnifiedYPosition, [1176](#)
- CEGUI::WindowProperties::VerticalAlignment, [1191](#)
- CEGUI::WindowProperties::Visible, [1201](#)
- CEGUI::WindowProperties::WantsMultiClickEvents, [1203](#)
- CEGUI::WindowProperties::WindowRenderer, [1324](#)
- CEGUI::WindowProperties::ZOrderChangeEnabled, [1340](#)
- setAlpha
  - CEGUI::ColourRect, [139](#)
  - CEGUI::Window, [1269](#)
- setAlwaysOnTop
  - CEGUI::Window, [1263](#)
- setArea
  - CEGUI::NamedArea, [704](#)
  - CEGUI::Window, [1276](#)
- setArea\_impl
  - CEGUI::Window, [1296](#)
- setAreaPropertySource
  - CEGUI::ComponentArea, [179](#)
- setArmed
  - CEGUI::ComboDropList, [173](#)
- setAutoArmEnabled
  - CEGUI::ComboDropList, [174](#)
- setAutoRepeatDelay
  - CEGUI::Window, [1271](#)
- setAutoRepeatRate
  - CEGUI::Window, [1272](#)
- setAutoResizeEnabled
  - CEGUI::ItemListBase, [497](#)
- setAutoScalingEnabled
  - CEGUI::Imageset, [463](#)
- setBackgroundHorizontalFormatting
  - CEGUI::FrameComponent, [364](#)
- setBackgroundVerticalFormatting
  - CEGUI::FrameComponent, [363](#)
- setBaseDimension
  - CEGUI::Dimension, [212](#)
- setBottomAlpha
  - CEGUI::ColourRect, [139](#)
- setButtonLocation
  - CEGUI::TreeItem, [1154](#)
- setCaratIndex
  - CEGUI::Combobox, [165](#)
  - CEGUI::Editbox, [262](#)
  - CEGUI::MultiLineEditbox, [687](#)
- setClickable
  - CEGUI::ListHeaderSegment, [578](#)
- setClickStep
  - CEGUI::Slider, [917](#)
- setClippedByParent
  - CEGUI::Window, [1265](#)
- setClippedToDisplay
  - CEGUI::StateImagery, [961](#)
- setClipperWindow
  - CEGUI::ClippedContainer, [133](#)
- setCloseButtonEnabled
  - CEGUI::FrameWindow, [376](#)
- setColours
  - CEGUI::ColourRect, [141](#)
  - CEGUI::FalagardComponentBase, [302](#)
- setColoursPropertyIsColourRect
  - CEGUI::FalagardComponentBase, [303](#)
- setColoursPropertySource
  - CEGUI::FalagardComponentBase, [302](#)
- setColumnDraggingEnabled
  - CEGUI::ListHeader, [564](#)
- setColumnHeaderWidth
  - CEGUI::MultiColumnList, [669](#)

- setColumnSizingEnabled
  - CEGUI::ListHeader, 564
- setColumnWidth
  - CEGUI::ListHeader, 568
- setComponentArea
  - CEGUI::FalagardComponentBase, 302
- setConstraintArea
  - CEGUI::MouseCursor, 628
- setContentArea
  - CEGUI::ScrolledContainer, 865
- setContentPaneArea
  - CEGUI::ScrollablePane, 838
- setContentPaneAutoSized
  - CEGUI::ScrollablePane, 838
  - CEGUI::ScrolledContainer, 864
- setCurrentValue
  - CEGUI::Slider, 917
  - CEGUI::Spinner, 953
- setData
  - CEGUI::RawDataContainer, 774
- setDefaultFont
  - CEGUI::System, 1046
- setDefaultMouseCursor
  - CEGUI::System, 1049, 1050
- setDefaultResourceGroup
  - CEGUI::Font, 324
  - CEGUI::ImageSet, 464
  - CEGUI::ResourceProvider, 802
  - CEGUI::Scheme, 816
  - CEGUI::ScriptModule, 829
  - CEGUI::WidgetLookManager, 1220
  - CEGUI::WindowManager, 1317
- setDefaultTooltip
  - CEGUI::System, 1052, 1053
- setDefaultXMLParserName
  - CEGUI::System, 1054
- setDestroyedByParent
  - CEGUI::Window, 1263
- setDimensionOperator
  - CEGUI::BaseDim, 104
- setDimensionType
  - CEGUI::Dimension, 212
- setDisabled
  - CEGUI::ListboxItem, 543
  - CEGUI::TreeItem, 1152
- setDisplayTime
  - CEGUI::Tooltip, 1118
- setDistributesCapturedInputs
  - CEGUI::Window, 1272
- setDocumentSize
  - CEGUI::Scrollbar, 855
- setDragAlpha
  - CEGUI::DragContainer, 231
- setDragCursorImage
  - CEGUI::DragContainer, 232
- setDragDropTarget
  - CEGUI::Window, 1284
- setDraggingEnabled
  - CEGUI::DragContainer, 230
  - CEGUI::Titlebar, 1107
- setDragMovingEnabled
  - CEGUI::FrameWindow, 377
  - CEGUI::ListHeaderSegment, 577
- setEnabled
  - CEGUI::Window, 1263
- setEWSizingCursorImage
  - CEGUI::FrameWindow, 378, 379
- setFadeInTime
  - CEGUI::PopupMenu, 730
- setFadeOutTime
  - CEGUI::PopupMenu, 731
- setFadeTime
  - CEGUI::Tooltip, 1119
- setFalagardType
  - CEGUI::Window, 1283
- setFont
  - CEGUI::ListboxTextItem, 548
  - CEGUI::TextComponent, 1086
  - CEGUI::TreeItem, 1148
  - CEGUI::Window, 1265, 1266
- setFontPropertySource
  - CEGUI::TextComponent, 1088
- setFrameEnabled
  - CEGUI::FrameWindow, 375
- setGroupID
  - CEGUI::RadioButton, 772
- setGUISheet
  - CEGUI::System, 1047
- setHeight
  - CEGUI::Window, 1278
- setHorizontalAlignment
  - CEGUI::Window, 1274
- setHorizontalFormatting
  - CEGUI::ImageryComponent, 446
  - CEGUI::TextComponent, 1087
- setHorizontalOverlapSize
  - CEGUI::ScrollablePane, 839
- setHorizontalScrollPosition
  - CEGUI::ScrollablePane, 840
- setHorizontalStepSize
  - CEGUI::ScrollablePane, 839
- setHorzFormattingPropertySource
  - CEGUI::FalagardComponentBase, 303
- setHorzFree
  - CEGUI::Thumb, 1102
- setHorzRange
  - CEGUI::Thumb, 1102
- setHotTracked

- CEGUI::Thumb, [1101](#)
- setHoverTime
  - CEGUI::Tooltip, [1119](#)
- setID
  - CEGUI::ListboxItem, [542](#)
  - CEGUI::TreeItem, [1151](#)
  - CEGUI::Window, [1265](#)
- setImage
  - CEGUI::FrameComponent, [364](#), [365](#)
  - CEGUI::ImageryComponent, [445](#)
  - CEGUI::MouseCursor, [627](#)
- setImagePropertySource
  - CEGUI::ImageryComponent, [447](#)
- setInheritsAlpha
  - CEGUI::Window, [1269](#)
- setInheritsTooltipText
  - CEGUI::Window, [1273](#)
- setItem
  - CEGUI::MultiColumnList, [664](#)
- setItemSelectState
  - CEGUI::Combobox, [168](#), [169](#)
  - CEGUI::Listbox, [531](#), [532](#)
  - CEGUI::MultiColumnList, [668](#)
  - CEGUI::Tree, [1137](#)
- setLeftAlpha
  - CEGUI::ColourRect, [140](#)
- setLogFilename
  - CEGUI::DefaultLogger, [208](#)
  - CEGUI::Logger, [585](#)
- setLoggingLevel
  - CEGUI::Logger, [585](#)
- setLookNFeel
  - CEGUI::Tree, [1138](#)
  - CEGUI::Window, [1274](#)
- setMaskCodePoint
  - CEGUI::Editbox, [262](#)
- setMasterColours
  - CEGUI::ImagerySection, [452](#)
- setMasterColoursPropertyIsColourRect
  - CEGUI::ImagerySection, [452](#)
- setMasterColoursPropertySource
  - CEGUI::ImagerySection, [452](#)
- setMaximumValue
  - CEGUI::Spinner, [954](#)
- setMaxSize
  - CEGUI::Window, [1279](#)
- setMaxTextLength
  - CEGUI::Combobox, [166](#)
  - CEGUI::Editbox, [263](#)
  - CEGUI::MultiLineEditbox, [688](#)
- setMaxValue
  - CEGUI::Slider, [916](#)
- setMinimumValue
  - CEGUI::Spinner, [954](#)
- setMinSize
  - CEGUI::Window, [1279](#)
- setModalState
  - CEGUI::Window, [1275](#)
- setModalTarget
  - CEGUI::System, [1053](#)
- setMouseAutoRepeatEnabled
  - CEGUI::Window, [1271](#)
- setMouseCursor
  - CEGUI::Window, [1270](#)
- setMouseMoveScaling
  - CEGUI::System, [1052](#)
- setMousePassThroughEnabled
  - CEGUI::Window, [1283](#)
- setMultiClickTimeout
  - CEGUI::System, [1048](#)
- setMultiClickToleranceAreaSize
  - CEGUI::System, [1049](#)
- setMultiselectEnabled
  - CEGUI::Listbox, [530](#)
  - CEGUI::Tree, [1136](#)
- setMutedState
  - CEGUI::EventSet, [283](#)
- setNativeResolution
  - CEGUI::Font, [321](#)
  - CEGUI::Imageset, [463](#)
- setNESWSizingCursorImage
  - CEGUI::FrameWindow, [379](#), [380](#)
- setNominatedSelectionColumn
  - CEGUI::MultiColumnList, [665](#)
- setNominatedSelectionColumnID
  - CEGUI::MultiColumnList, [665](#)
- setNominatedSelectionRow
  - CEGUI::MultiColumnList, [666](#)
- setNSSizingCursorImage
  - CEGUI::FrameWindow, [378](#), [379](#)
- setNWSESizingCursorImage
  - CEGUI::FrameWindow, [378](#), [380](#)
- setOperand
  - CEGUI::BaseDim, [104](#)
- setOverlapSize
  - CEGUI::Scrollbar, [856](#)
- setOverrideColours
  - CEGUI::SectionSpecification, [876](#)
- setOverrideColoursPropertyIsColourRect
  - CEGUI::SectionSpecification, [877](#)
- setOverrideColoursPropertySource
  - CEGUI::SectionSpecification, [876](#)
- setOwnerWindow
  - CEGUI::ListboxItem, [543](#)
  - CEGUI::TreeItem, [1153](#)
- setPageSize
  - CEGUI::Scrollbar, [855](#)
- setParent

- CEGUI::Window, 1295
- setPixelDragThreshold
  - CEGUI::DragContainer, 231
- setPopupMenu
  - CEGUI::MenuItem, 616
- setPopupMenu\_impl
  - CEGUI::MenuItem, 618
- setPosition
  - CEGUI::MouseCursor, 628
  - CEGUI::Window, 1277
- setPrefix
  - CEGUI::Window, 1265
- setProgress
  - CEGUI::ProgressBar, 737
- setProperty
  - CEGUI::PropertySet, 765
- setQueueingEnabled
  - CEGUI::Renderer, 794
- setReadOnly
  - CEGUI::Combobox, 165
  - CEGUI::Editbox, 261
  - CEGUI::MultiLineEditbox, 687
- setRenderControlPropertySource
  - CEGUI::SectionSpecification, 877
- setRestoreCapture
  - CEGUI::Window, 1268
- setRightAlpha
  - CEGUI::ColourRect, 140
- setRiseOnClickEnabled
  - CEGUI::Window, 1274
- setRollupEnabled
  - CEGUI::FrameWindow, 376
- setRowID
  - CEGUI::MultiColumnList, 670
- setScriptingModule
  - CEGUI::System, 1050
- setScrollPosition
  - CEGUI::Scrollbar, 856
- setSegmentOffset
  - CEGUI::ListHeader, 568
- setSelectable
  - CEGUI::ItemEntry, 483
- setSelected
  - CEGUI::Checkbox, 120
  - CEGUI::ItemEntry, 483
  - CEGUI::ListboxItem, 543
  - CEGUI::RadioButton, 772
  - CEGUI::TreeItem, 1152
- setSelection
  - CEGUI::Combobox, 166
  - CEGUI::Editbox, 262
  - CEGUI::MultiLineEditbox, 687
- setSelectionBrushImage
  - CEGUI::ListboxItem, 544, 545
  - CEGUI::TreeItem, 1154
- setSelectionColours
  - CEGUI::ListboxItem, 544
  - CEGUI::TreeItem, 1153, 1154
- setSelectionMode
  - CEGUI::MultiColumnList, 665
- setShowHorzScrollbar
  - CEGUI::Combobox, 168
  - CEGUI::Listbox, 531
  - CEGUI::MultiColumnList, 667
  - CEGUI::ScrollablePane, 837
  - CEGUI::Tree, 1137
- setShowVertScrollbar
  - CEGUI::Combobox, 168
  - CEGUI::Listbox, 531
  - CEGUI::MultiColumnList, 667
  - CEGUI::MultiLineEditbox, 688
  - CEGUI::ScrollablePane, 837
  - CEGUI::Tree, 1136
- setSingleClickEnabled
  - CEGUI::Combobox, 165
- setSingleClickTimeout
  - CEGUI::System, 1048
- setSize
  - CEGUI::RawDataContainer, 774
  - CEGUI::Window, 1278
- setSizeingBorderThickness
  - CEGUI::FrameWindow, 376
- setSizeingEnabled
  - CEGUI::FrameWindow, 375
  - CEGUI::ListHeaderSegment, 577
- setSortCallback
  - CEGUI::ItemListBase, 498
- setSortColumn
  - CEGUI::ListHeader, 564
  - CEGUI::MultiColumnList, 666
- setSortColumnByID
  - CEGUI::MultiColumnList, 667
- setSortColumnFromID
  - CEGUI::ListHeader, 564
- setSortDirection
  - CEGUI::ListHeader, 563
  - CEGUI::ListHeaderSegment, 577
  - CEGUI::MultiColumnList, 666
- setSortingEnabled
  - CEGUI::Combobox, 167
  - CEGUI::Listbox, 530
  - CEGUI::ListHeader, 563
  - CEGUI::Tree, 1136
- setSortMode
  - CEGUI::ItemListBase, 498
- setSortSegment
  - CEGUI::ListHeader, 563
- setSourceDimension

- CEGUI::ImageDim, 442
- CEGUI::WidgetDim, 1207
- setSourceImage
  - CEGUI::ImageDim, 442
- setStepSize
  - CEGUI::ProgressBar, 737
  - CEGUI::Scrollbar, 855
  - CEGUI::Spinner, 953
- setTabPanePosition
  - CEGUI::TabControl, 1067
- setTargetWindow
  - CEGUI::Tooltip, 1118
- setText
  - CEGUI::ListboxItem, 542
  - CEGUI::TextComponent, 1085
  - CEGUI::TreeItem, 1151
  - CEGUI::Window, 1265
- setTextColours
  - CEGUI::ListboxTextItem, 548, 549
  - CEGUI::TreeItem, 1149
- setTextInputMode
  - CEGUI::Spinner, 954
- setTextMasked
  - CEGUI::Editbox, 261
- setTextPropertySource
  - CEGUI::TextComponent, 1088
- setTexture
  - CEGUI::Imageset, 464
- setTitleBarEnabled
  - CEGUI::FrameWindow, 376
- setTooltip
  - CEGUI::Window, 1272
- setTooltipText
  - CEGUI::Window, 1273
- setTooltipType
  - CEGUI::Window, 1273
- setTopAlpha
  - CEGUI::ColourRect, 139
- setUnifiedConstraintArea
  - CEGUI::MouseCursor, 628
- setUserColumnDraggingEnabled
  - CEGUI::MultiColumnList, 669
- setUserColumnSizingEnabled
  - CEGUI::MultiColumnList, 669
- setUserData
  - CEGUI::ListboxItem, 542
  - CEGUI::TreeItem, 1152
  - CEGUI::Window, 1270
- setUserSortControlEnabled
  - CEGUI::MultiColumnList, 669
- setUserString
  - CEGUI::Window, 1275
- setUsingOverrideColours
  - CEGUI::SectionSpecification, 876
- setValidationString
  - CEGUI::Combobox, 165
  - CEGUI::Editbox, 262
- setVertFormattingPropertySource
  - CEGUI::FalagardComponentBase, 303
- setVertFree
  - CEGUI::Thumb, 1101
- setVerticalAlignment
  - CEGUI::Window, 1274
- setVerticalFormatting
  - CEGUI::ImageryComponent, 445
  - CEGUI::TextComponent, 1086
- setVerticalOverlapSize
  - CEGUI::ScrollablePane, 840
- setVerticalScrollPosition
  - CEGUI::ScrollablePane, 841
- setVerticalStepSize
  - CEGUI::ScrollablePane, 840
- setVertRange
  - CEGUI::Thumb, 1102
- setVisible
  - CEGUI::MouseCursor, 629
  - CEGUI::Window, 1264
- setWantsMultiClickEvents
  - CEGUI::Window, 1271
- setWidgetName
  - CEGUI::WidgetDim, 1207
- setWidth
  - CEGUI::Window, 1278
- setWindowRenderer
  - CEGUI::Window, 1283
- setWordWrapping
  - CEGUI::MultiLineEditbox, 688
- setXPosition
  - CEGUI::Window, 1277
- setYPosition
  - CEGUI::Window, 1277
- setZOrderingEnabled
  - CEGUI::Window, 1271
- Shift
  - CEGUI, 49
- show
  - CEGUI::MouseCursor, 629
  - CEGUI::Window, 1264
- showDropList
  - CEGUI::Combobox, 164
- signalRedraw
  - CEGUI::System, 1047
- size
  - CEGUI::String, 982
- sizeSelf
  - CEGUI::Tooltip, 1119
- sizeToContent
  - CEGUI::ItemListBase, 497



- sizeToContent\_impl
  - CEGUI::ItemListBase, 499
- SizingBottom
  - CEGUI::FrameWindow, 374
- SizingBottomLeft
  - CEGUI::FrameWindow, 374
- SizingBottomRight
  - CEGUI::FrameWindow, 374
- SizingLeft
  - CEGUI::FrameWindow, 374
- SizingLocation
  - CEGUI::FrameWindow, 373
- SizingNone
  - CEGUI::FrameWindow, 374
- SizingRight
  - CEGUI::FrameWindow, 374
- SizingTop
  - CEGUI::FrameWindow, 374
- SizingTopLeft
  - CEGUI::FrameWindow, 374
- SizingTopRight
  - CEGUI::FrameWindow, 374
- SortDirection
  - CEGUI::ListHeaderSegment, 576
- sortList
  - CEGUI::ItemListBase, 498
- Standard
  - CEGUI, 48
- StateImagery
  - CEGUI::StateImagery, 960
- step
  - CEGUI::ProgressBar, 737
- String
  - CEGUI::String, 978–981
- subscribe
  - CEGUI::Event, 275
- subscribeEvent
  - CEGUI::EventSet, 281, 282
  - CEGUI::ScriptModule, 828
- subscribeScriptedEvent
  - CEGUI::EventSet, 282
- substr
  - CEGUI::String, 1032
- swap
  - CEGUI, 53
  - CEGUI::String, 995
- System
  - CEGUI::System, 1045
- SystemKey
  - CEGUI, 49
- testClassName
  - CEGUI::Window, 1258
- testClassName\_impl
  - CEGUI::ButtonBase, 112
  - CEGUI::Checkbox, 121
  - CEGUI::ClippedContainer, 133
  - CEGUI::Combobox, 169
  - CEGUI::ComboDropList, 174
  - CEGUI::DragContainer, 233
  - CEGUI::Editbox, 264
  - CEGUI::FrameWindow, 383
  - CEGUI::GroupBox, 402
  - CEGUI::GUISheet, 410
  - CEGUI::ItemEntry, 484
  - CEGUI::ItemListBase, 500
  - CEGUI::ItemListbox, 507
  - CEGUI::Listbox, 534
  - CEGUI::ListHeader, 569
  - CEGUI::ListHeaderSegment, 579
  - CEGUI::Menubar, 608
  - CEGUI::MenuBase, 612
  - CEGUI::MenuItem, 619
  - CEGUI::MultiColumnList, 671
  - CEGUI::MultiLineEditbox, 689
  - CEGUI::PopupMenu, 732
  - CEGUI::ProgressBar, 738
  - CEGUI::PushButton, 768
  - CEGUI::RadioButton, 772
  - CEGUI::ScrollablePane, 843
  - CEGUI::Scrollbar, 857
  - CEGUI::ScrolledContainer, 865
  - CEGUI::ScrolledItemListBase, 870
  - CEGUI::Slider, 918
  - CEGUI::Spinner, 955
  - CEGUI::TabButton, 1061
  - CEGUI::TabControl, 1070
  - CEGUI::Thumb, 1102
  - CEGUI::Titlebar, 1109
  - CEGUI::Tooltip, 1120
  - CEGUI::Tree, 1140
  - CEGUI::Window, 1295
- text
  - CEGUI::XMLSerializer, 1338
- TextFormatting
  - CEGUI, 49
- TextInputMode
  - CEGUI::Spinner, 952
- TipState
  - CEGUI::Tooltip, 1117
- togglePopupMenu
  - CEGUI::MenuItem, 616
- toggleRollup
  - CEGUI::FrameWindow, 376
- TopLeftToBottomRight
  - CEGUI, 49
- trimLeadingChars
  - CEGUI::TextUtils, 1096

- trimTrailingChars
  - CEGUI::TextUtils, [1097](#)
- undefineAllImages
  - CEGUI::Imageset, [459](#)
- undefineImage
  - CEGUI::Imageset, [458](#)
- UnifiedDim
  - CEGUI::UnifiedDim, [1160](#)
- UnknownObjectException
  - CEGUI::UnknownObjectException, [1177](#)
- unloadAllSchemes
  - CEGUI::SchemeManager, [820](#)
- unloadRawDataContainer
  - CEGUI::ResourceProvider, [802](#)
- unloadResources
  - CEGUI::Scheme, [815](#)
- unloadScheme
  - CEGUI::SchemeManager, [820](#)
- update
  - CEGUI::Window, [1282](#)
- updateImageScalingFactors
  - CEGUI::Imageset, [465](#)
- updateInternalState
  - CEGUI::ButtonBase, [111](#)
  - CEGUI::MenuItem, [618](#)
- updateSelf
  - CEGUI::PopupMenu, [731](#)
  - CEGUI::Tooltip, [1122](#)
  - CEGUI::Window, [1294](#)
- utf8\_stream\_len
  - CEGUI::String, [990](#)
- VA\_BOTTOM
  - CEGUI, [50](#)
- VA\_CENTRE
  - CEGUI, [50](#)
- VA\_TOP
  - CEGUI, [50](#)
- validateWindowRenderer
  - CEGUI::Editbox, [264](#)
  - CEGUI::ItemEntry, [484](#)
  - CEGUI::ItemListBase, [500](#)
  - CEGUI::Listbox, [535](#)
  - CEGUI::ListHeader, [570](#)
  - CEGUI::MultiColumnList, [671](#)
  - CEGUI::MultiLineEditbox, [690](#)
  - CEGUI::ScrollablePane, [843](#)
  - CEGUI::Scrollbar, [858](#)
  - CEGUI::Slider, [918](#)
  - CEGUI::TabControl, [1071](#)
  - CEGUI::Tooltip, [1120](#)
  - CEGUI::Window, [1295](#)
- VerticalAlignment
  - CEGUI, [50](#)
- VerticalFormatting
  - CEGUI, [50](#)
- VerticalTextFormatting
  - CEGUI, [50](#)
- VF\_BOTTOM\_ALIGNED
  - CEGUI, [50](#)
- VF\_CENTRE\_ALIGNED
  - CEGUI, [50](#)
- VF\_STRETCHED
  - CEGUI, [50](#)
- VF\_TILED
  - CEGUI, [50](#)
- VF\_TOP\_ALIGNED
  - CEGUI, [50](#)
- VTF\_BOTTOM\_ALIGNED
  - CEGUI, [50](#)
- VTF\_CENTRE\_ALIGNED
  - CEGUI, [50](#)
- VTF\_TOP\_ALIGNED
  - CEGUI, [50](#)
- wantsMultiClickEvents
  - CEGUI::Window, [1256](#)
- Warnings
  - CEGUI, [48](#)
- WidgetDim
  - CEGUI::WidgetDim, [1207](#)
- Window
  - CEGUI::Window, [1245](#)
- WindowManager
  - CEGUI::WindowManager, [1312](#)
- WindowRenderer
  - CEGUI::WindowRenderer, [1321](#)
- WindowRendererFactory
  - CEGUI::WindowRendererFactory, [1325](#)
- windowToScreen
  - CEGUI::CoordConverter, [195](#), [196](#)
- windowToScreenX
  - CEGUI::CoordConverter, [194](#)
- windowToScreenY
  - CEGUI::CoordConverter, [194](#), [195](#)
- WordWrapCentred
  - CEGUI, [50](#)
- WordWrapJustified
  - CEGUI, [50](#)
- WordWrapLeftAligned
  - CEGUI, [50](#)
- WordWrapRightAligned
  - CEGUI, [50](#)
- writeColoursXML
  - CEGUI::FalagardComponentBase, [304](#)
- writeFontToStream
  - CEGUI::FontManager, [334](#)



- writeHorzFormatXML
  - CEGUI::FalagardComponentBase, [304](#)
- writeImagesetToStream
  - CEGUI::ImagesetManager, [471](#)
- writeVertFormatXML
  - CEGUI::FalagardComponentBase, [304](#)
- writeWidgetLookSeriesToStream
  - CEGUI::WidgetLookManager, [1220](#)
- writeWidgetLookToStream
  - CEGUI::WidgetLookManager, [1220](#)
- writeWindowLayoutToStream
  - CEGUI::WindowManager, [1315](#), [1316](#)
- writeXMLAttributes
  - CEGUI::PropertyDefinitionBase, [753](#)
  - CEGUI::PropertyLinkDefinition, [761](#)
- writeXMLElementType
  - CEGUI::PropertyDefinition, [749](#)
  - CEGUI::PropertyDefinitionBase, [752](#)
  - CEGUI::PropertyLinkDefinition, [761](#)
- writeXMLToStream
  - CEGUI::BaseDim, [104](#)
  - CEGUI::ComponentArea, [178](#)
  - CEGUI::Dimension, [213](#)
  - CEGUI::Font, [317](#)
  - CEGUI::FrameComponent, [365](#)
  - CEGUI::Image, [437](#)
  - CEGUI::ImageryComponent, [446](#)
  - CEGUI::ImagerySection, [452](#)
  - CEGUI::Imageset, [463](#)
  - CEGUI::LayerSpecification, [520](#)
  - CEGUI::NamedArea, [704](#)
  - CEGUI::Property, [747](#)
  - CEGUI::PropertyDefinitionBase, [752](#)
  - CEGUI::PropertyInitialiser, [758](#)
  - CEGUI::SectionSpecification, [877](#)
  - CEGUI::StateImagery, [961](#)
  - CEGUI::TextComponent, [1087](#)
  - CEGUI::WidgetComponent, [1205](#)
  - CEGUI::WidgetLookFeel, [1215](#)
  - CEGUI::Window, [1282](#)
  - CEGUI::WindowProperties::LookNFeel, [588](#)
  - CEGUI::WindowProperties::WindowRenderer,  
[1324](#)
- writeXMLToStream\_impl
  - CEGUI::Font, [317](#)
- X1Mouse
  - CEGUI, [49](#)
- X2Mouse
  - CEGUI, [49](#)
- XMLSerializer
  - CEGUI::XMLSerializer, [1337](#)