



**Wide Studio
Programming Guide**

Contents

A	The event procedures	1
1	How to access to the gui instaces	1
1.1	Accessing of the instance by the parameter of the event procedure	1
1.2	Accessing of the instance by using the instance management	1
1.3	Accessing of the global instance directly	2
2	How to access a value of the property of the instance	3
2.1	How to get a value of the properties	3
2.2	Setting of property	4
2.3	Updating the instance to refrect the changed value	4
3	How to cast the WSCbase* into the specified class	5
4	How to access to the method of the specified class	7
5	How to get the parent instace	8
5.1	How to get the parent top level window	8
6	How to get the children of the instace	9
6.1	How to get the child by the specified name	9
6.2	How to get the children	9
6.3	How to get all of the children recursively	10
6.4	How to get all of the children of the parent application window	10
7	How to execute the event procedures of the instances	11
7.1	How to execute the event procedures by the specified name	11
7.2	How to execute the event procedures by the specified trigger	11
8	How to draw the instances	12
9	How to update the instance	12
9.1	How to draw the instance	12
10	How to move the position of the instances	14
11	How to create/delete the instances	15
11.1	How to create the instance	15
11.2	How to delete the instance	15
12	How to use of the timer	16
13	How to execute the procedure after an interval	16
14	How to execute the procedure after in cycles	17
15	How to use the global key hook function	18

16	How to select the key events on the input field	19
17	How to add the event procedure on the programs	20
18	How to access to the arrayed instance.	21
19	How to indicate a dialog on the WSEV_EXIT event procedure	22
20	How to examine which mouse button is pressed	24
B	The sample of the event procedures	25
1	The sample of the event procedures for WSCvlabel	25
1.1	Making the WSCvlabel instance click-able	25
1.2	Making the WSCvlabel instance select-able	26
1.3	Making the WSCvlabel instance highlight-able	27
1.4	Making a group of selectable WSCvlabel instances	28
2	The sample of the event procedures for WSCvifield	30
2.1	Executing some event procedures by return key	30
2.2	Clearing the last input string on starting of next input	30
3	Automatic geometry adjustment with the anchors	33
4	The pulldown menu and the menu area	34
4.1	What is the menu area	34
4.2	Try to use the pulldown menu	34
4.3	Notice of the pulldown menu	35
5	The list	36
5.1	Setting of the items by the method	36
5.2	Setting of the items by the property	36
5.3	Setting of the items from the file directly	37
5.4	Setting of the items from the other instance	38
6	The verbose list	40
6.1	Setting of the items by the method	40
6.2	Setting of the items by the property	40
6.3	Setting of the items from the file directly	41
6.4	Setting of the items from the other instance	42
7	The tree list	43
7.1	Setting of the items by the method	43
7.2	Setting of the items by the property	44
7.3	Setting of the items from the file directly	44
7.4	Setting of the items from the other instance	45

8	The user dialog	47
8.1	How to make a simple user dialog	47
8.2	Controlling to indicate the user dialog	47
9	The file selection dialog	52
9.1	Indication of the file selection dialog	52
10	The scrolled form	54
10.1	How to use the virtual scrolling	54
11	The separated form	55
12	How to set the width of the separated area	55
13	The drawing area	56
13.1	How to draw pictures on the drawing area	56
13.2	How to draw images(JPG,BMP) on the drawing area	57
14	The indexed form	58
15	The balloon help	59
15.1	How to indicate the balloon help	59
16	The timer	60
16.1	How to use the timer	60
17	The wizard dialog	61
18	The controlling of the position of the instances by the offset	62
18.1	The offset of X,Y coordinate	62
18.2	The scaling offset of the size of the instance	62
19	The memory device class	63
19.1	How to create the memory device class	63
20	Network communication using TCP/IP	66
20.1	How to use network communication using TCP sockets	66
20.2	How to use a broadcast network communication using UDP socket	69
21	Database access using the database class	71
21.1	Database access through ODBC	71
21.2	Database access through PostgreSQL interface	72
21.3	Creating the table	73
21.4	Store data in the table	73
21.5	Referring data on the table	74
C	The user defined classes	76

1	How to access to the member instances	76
1.1	How to access to the member instances in the class event procedures	76
1.2	How to access to the member instances in the method	76
D	The store function	77
1	How to load the stored application window directly from the program	77
1.1	How to load the stored application window directly from the program	77
1.2	How to load the stored partial application window directly from the program	78
2	How to delete the loaded application window	79
2.1	How to delete the loaded application window	79
2.2	How to delete the loaded partial application window	79
E	The remote instance function	80
1	Accessing a remote instance	80
1.1	Accessing a remote instance	80
1.2	Casting a remote instance	80
F	Samples and demonstrations	82
1	Sample:1 (Hello World)	82
2	Sample:2(Various kinds of classes)	84
3	Sample:3 (label)	86
4	Sample:4 (A calculator)	87

A The event procedures

1 How to access to the gui instaces

In the event procedures, It Is the most fundamental to access to the instances. So I will explain the various kinds of accessing the instance in this section.

- Accessing of the instance by the parameter of the event procedure
In the event procedure which the instance has, we use the parameter of the procedure to access the instance,
- Accessing of the instance by using the instance management
We use the instance management to access the instance except it which the event procedure is set.
- Accessing of the global instance directly
You can use the global instance to access directly. it is easy to access and is good performance, but it makes the possibility of porting worse.

1.1 Accessing of the instance by the parameter of the event procedure

```
void event_procedure(WSCbase* object){
    //accessing the instance.
    object->setProperty(WSNlabelString,"HELLO WORLD");
}
```

the pointer: object is the instace which is the client of the event procedure that fired. the type is WSCbase*, so you can access the method of WSCbase, but you must cast the pointer by the method:cast() to access the other subclass: see the capter of casting sub class.

1.2 Accesing of the instance by using the instance management

The instance management returns the instance by name.

The instance management class	The instance acquisition method
WSCbaseList	WSCbaseList* WSGIappObjectList()

To access of the instace as follows:

```
#include "WSCbaseList.h" //to access WSGIappObjectList()
...
void event_procedure(WSCbase* object){

    //Acquisition of the instance(1)
    char* class_name = "WSCvlabel"; //Seek from the instance of WSCvlabel class
    char* obj_name   = "newvlab_001"; //the instance name is newvlab_001
    WSCbase* object = WSGIappObjectList()->getInstance(class_name,obj_name);
    object->setProperty(WSNlabelString,"HELLO WORLD");

    //Acquisition of the instance(2)
```

```

char* class_name2 = "WSCbase"; //Seek from all the instances.
char* obj_name2   = "newvlab_002"; //the instace name is newvlab_002
WSCbase* object2 = WSGIappObjectList()->getInstance(class_name2,obj_name2);
object2->setProperty(WSNlabelString,"HELLO WORLD");

```

The variable: obeit is the seeked instance by the specified class name and the specified instance name. If you do not want to specify the class name, you can pass "WSCbase", it seeks from all the instaces.

1.3 Accesing of the global instance directly

Make the instance globa,you can access it directly. See the capter [Setting of global instance] of Application Builder User's Guide to make it global.

```

#include "WSCvlabel.h" //to access WSCvlabel class
...
void event_procedure(WSCbase* object){
    //extern of the global instance
    extern WSCvlabel* newvlab_001;

    //Access the instance of WSCvlabel*: newvlab_001
    newvlab_001->setProperty(WSNlabelString,"HELLO WORLD");
}

```

2 How to access a value of the property of the instance

In the event program, you can access the property of the instance by the method: `get/setProperty()`.

the access method	Description
<code>getProperty</code>	get a value of the specified property
<code>setProperty</code>	set a value to the specified property

2.1 How to get a value of the properties

You can get a value of the property by the method of `WSCbase`: `getProerty()`.

```
void event_procedure(WSCbase* object){

    //To get a value of the property: WSNx by string type.
    WSCstring x = object->getProperty(WSNx);
    printf("x=%s\n", (char*)x);

    //To get a value of the property: WSNy by short type.
    short y = object->getProperty(WSNy);

}
```

In the example of `WSNx`, it gets a value by string type. The `WSCstring` type manages the buffer of the string automatically, so there is no need to manage it by programmer.

In the example of `WSNy`, it gets a value by short type. The method: `getProperty()` returns a value by `WSCvariant` type. The `WSCvariant` type can convert the various kind of type automatically. The following example is a some conversion short type into string type.

```
void cbop(WSCbase* object){

    //To get a value of the property: WSNx by string type.
    WSCstring x = object->getProperty(WSNx);
    printf("x=%s\n", (char*)x);

    //To get a value of the property: WSNy by short type.
    short y = object->getProperty(WSNy);

    //convert into string..
    WSCvariant stry = y;
    printf("y=%s\n", (char*)stry);
    //convert into double.
    printf("y=%f\n", (double)stry);

}
```

Notice: You can not get the value by `char*`, if you use it, the pointer will be junk pointer when the method of `WSCbase`: `getProperty()` will be done, because the returned value is auto variable of the `WSCvariant`, so the `WSCvariant` instance and its internal buffer for string is destroyed when the method is done.

If you want to get the char* pointer, see the following program.

```
void event_procedure(WSCbase* object){

    //To get a value of the property: WSNlabelString by char pointer.
    //Not good! the poiter string will be junc pointer!
    char* string = object->getProperty(WSNlabelString);

    //To get a value of the property: WSNlabelString by char pointer.
    //Good example. The string1 (WSCstirng instance) keeps the string buffer.
    WSCstring string1
    string1 = object->getProperty(WSNlabelString);
    char* str = (char*)string1;

}
```

2.2 Setting of property

You can set a value to the property by the method of WSCbase: setProerty().

```
void event_procedure(WSCbase* object){

    //To set a value to the property: WSNx by string type.
    char* x="100";
    object->setProperty(WSNx,x);

    //To set a value to the property: WSNy by short type.
    short y=100;
    object->getProperty(WSNy,y);

}
```

In the example of WSNx, it sets a value by string type. In the example of WSNy, it sets a value by short type. The parameter of the method: setProperty() is WSCvariant type, so it converts the value automatically.

2.3 Updating the instance to refrect the changed value

Use the method: update(),draw(),redraw() to update the instance to refrect the canged value.

```
obj1->getProperty(WSNlabelString,"string1");
obj1->update(); //updates the instance
obj2->getProperty(WSNlabelString,"string2");
obj2->update(); //updates the instance
```

Usually, it updates automatically when the event procedure is done, but if you want to update right away,you can use the method: update(),draw(),redraw().

The method: update() or draw() updates the instance if needed, but redraw() updates compulsory.

3 How to cast the WSCbase* into the specified class

To access a method of some subclass, it requires that the pointer is subclass. So we must convert (downcast) the pointer of WSCbase* into the subclass with some method. I will explain the acquisition of the casted pointer in this chapter.

- Casting of the casted pointer with the method of WSCbase: cast()

The method of casting	Description
void* WSCbase::cast(char* className)	Returns the pointer of the specified class.

Usage of the method: WSCbase::cast() is as follows. In the following example, the pointer "object" contains a WSCvtoggle instance, but is a pointer of WSCbase*. and you want to access the WSCvtoggle method: getStatus() which returns the state of the toggle.

In c++ language, it is not allowed to downcast like WSCbase* to WSCvtoggle*. The method: WSCbase::cast() supports this.

```
#include "WSCvtoggle.h" //access WSCvtoggle class.
...

void cbop(WSCbase* object){
    //downcast WSCbase* to WSCvtoggle*.
    WSCvtoggle* tgl = (WSCvtoggle*)object->cast("WSCvtoggle");

    if (tgl == NULL){
        // it fails, because the pointer "object" is not a WSCvtoggle instance.
    }else{
        // it succeeds, the pointer "object" is a WSCvtoggle instance.
        // access the WSCvtoggle::getStatus()
        WSCbool status = tgl->getStatus();
    }
}
```

The method: WSCbase::cast() returns NULL, if the instance is not a instance of the specified class. if we use this specification well, we can examine the instance whether it is the specified class or not.

```
#include "WSCvbtn.h" //access WSCvbtn class.
#include "WSCvtoggle.h" //access WSCvtoggle class.
...

void cbop(WSCbase* object){
    //examine whether object is a WSCvlabel instance.
    WSCvlabel* btn = (WSCvlabel*)object->cast("WSCvlabel");

    //examine whether object is a WSCvtoggle instance.
    WSCvtoggle* toggle = (WSCvtoggle*)object->cast("WSCvtoggle");
```

```
if (btn == NULL){
    //it is not a WSCvbtn instance.
}else{
    //it is a WSCvbtn instance or inherits WSCvbtn class.
}
if (toggle == NULL){
    //it is not a WSCvtoggle instance.
}else{
    //it is a WSCvtoggle instance or inherits WSCvtoggle class.
}
}
```

If the event procedure is used by some instances of various classes, it is useful to switch the program.

4 How to access to the method of the specified class

In the chapter: [To cast the WSCbase* into the specified class], you can cast WSCbase* to the pointer of the specified class. Following example shows accessing of the method of WSClist class. WSClist class displays the list of some strings, and the WSClist::addString() method adds the specified string to the list.

```
#include "WSClist.h" //use WSClist class.
...
void some_function(...){
    //get the instance: name=list001, class=WSClist.
    WSCbase* object;
    object = (WSCbase*)WSGIappObjectList()->getInstance("WSClist","list001");

    //cast to WSClist pointer, and execute WSClist::addItem().
    WSClist* list = (WSClist*)object->cast("WSClist");
    list->addItem("sample string",0)
```

In the example, it gets the instance which name is "list001" and which class is WSClist from the instance management. The instance management returns WSCbase*, it casts WSCbase* to WSClist*, and executes the WSClist::addItem() method. In the top of the source, we must include the header file of the class.

5 How to get the parent instace

You can get the parent instace with the method: `WSCbase::getParent()`.

In the following sample, it acquires the parent instance which places the instace "object", and set `False` to the visibility of the parent.

```
void event_procedure(WSCbase* object){
    //get the parent instance.
    WSCbase* parent = object->getParent();
    //access the parent instance.
    parent->setVisible(False);
}
```

5.1 How to get the parent top level window

You can get the parent window instance with the method: `WSCbase::getParentWindow()`.

In the following sample, it acquires the parent window instance, and pops down the window. Sometimes this scene is found when closing window by a button instance.

```
void event_procedure(WSCbase* object){
    //get the parent window.
    WSCbase* win = object->getParent();
    //popdown the window.
    win->setVisible(False);
}
```

6 How to get the children of the instance

You can get the children which the instance contains with the following method.

The method	Description
WSCbase* getChildInstance(char*)	Returns the child by the instance name
WSClistData getChildren()	Returns the children
long getAllChildren(WSClistData&)	Returns recursively all of the children

6.1 How to get the child by the specified name

You can get the child by the specified name with the method: WSCbase::getChildInstance();

```
void event_procedure(WSCbase* object){
    //get the child which name is "newpbtn001"
    WSCbase* child = object->getChildInstance("newpbtn001");
    if (child != NULL){
        //the child exists.
        child->setVisible(True);
    }
}
```

In the example, WSCbase::getChildInstance() seeks the child which name is "newpbtn001" from the instance: "object" recursively, and returns it if finds or returns NULL if not.

6.2 How to get the children

You can get the children which the instance contains with the method: WSCbase::getChildren();

```
void event_procedure(WSCbase* object){
    //get the list of the children.
    WSClistData children = object->getChildren();
    //get the number of the children.
    long num = children.getNum();
    long i;
    for(i=0; i<num; i++){
        //get each child from the list.
        WSCbase* child = (WSCbase*)children[i];
        // WSCbase* child = (WSCbase*)children.getData(i); //same as children[i].
        //access the child.
        child->setVisible(False);
    }
}
```

The example shows acquisition of the children which the instance contains. The method: WSCbase::getChildren() returns the list: WSClistData which contains the children of the instance. You can access each child with the method getData(i), array operator[i] of the list, and cast void* to WSCbase*.

6.3 How to get all of the children recursively

You can get all of the children recursively with the method `WSCbase::getAllChildren()`.

```
void event_procedure(WSCbase* object){
    //the list for the return value of getAllChildren().
    WSClistData children;
    //get all of the children.
    object->getAllChildren(children);
    //get the number of children.
    long num = children.getNum();
    long i;
    for(i=0; i<num; i++){
        //get each child from the list.
        WSCbase* child = (WSCbase*)children[i];
        //access the child.
        child->setVisible(True);
    }
}
```

The example shows acquisition of all of the children. The difference from `getChildren()` is that it returns recursively all of the children of the instance, of the children,....

6.4 How to get all of the children of the parent application window

We call the method: `WSCbase::getAllChildren()` of the parent application window to get all of the children of it.

```
void cbop(WSCbase* object){
    //get the parent application window.
    WSCbase* win = object->getParentWindow();
    //a list for the return value.
    WSClistData children;
    //get all of the children of the application window.
    win->getAllChildren(children);
    //get the number of the children.
    long num = children.getNum();
    long i;
    for(i=0; i<num; i++){
        //get each child.
        WSCbase* child = (WSCbase*)children[i];
        //access the child.
        child->setVisible(False);
    }
}
```

7 How to execute the event procedures of the instances

You can execute the event procedures which the instances have with the method: WSCbase::execProcedure

Executing the event procedure	Description
execProcedure(char*)	Executing by name
execProcedure(long)	Executing by trigger

7.1 How to execute the event procedures by the specified name

You can execute the event procedures which the instances have by the specified name with the method: WSCbase::execProcedure(char*).

```
void event_procedure(WSCbase* object){
    //Execute the event procedures which name is "setup"
    object->execProcedure("setup");
}
```

If exists the event procedures of the instance: "object", which name is same as the specified one, It executes it, but not , does nothing.

7.2 How to execute the event procedures by the specified trigger

You can execute the event procedures which the instances have by the specified trigger with the method: WSCbase::execProcedure(int).

```
void event_procedure(WSCbase* object){
    // Execute the procedures which trigger is WSEV_ACTIVATE.
    object->execProcedure(WSEV_ACTIVATE);
}
```

If exists the event procedures of the instance: "object", which trigger is same as the specified one, It executes it, but not , does nothing.

8 How to draw the instances

You can use the following methods to draw the instances.

To control drawing	Description
setAbsoluteDraw(Boolean)	Sets the flag of forced drawing.
draw()	draws if needs.
redraw()	clears and draws
cdraw()	draws the instance and its children.
clear()	clears the instance.
update()	draws if needs.

9 How to update the instance

Usually, it executes updating the instance at the end of the event procedures. If you want to update at once, you can it with the method: update().

```
void event_procedure(WSCbase* object){
    //change a property.
    object->setProperty(WSNlabelString,"new text");
    //updating.
    object->update();
}
```

The method: update() updates the instance, if it needs to reflect the change of the properties.

9.1 How to draw the instance

There are following cases to draw the instances.

- Drawing compulsory
You make the flag of forced drawing "True" to draw compulsory, and call the method:draw(), because the method: draw() do not draw if it does not need.

```
object->setAbsoluteDraw(True);
object->draw();
```

- Drawing (ordinary)
Usually , we use the method: draw() to draw the instace. it has a good performance, because it do not draw if do not need.

```
object->draw();
```

- Clearing the instance (no exposed event)
You can clear the instace with the method: clear(), and draw it with draw(). The method: clear() does not creates the exposed event, so, if the other instance which are overlaped exists, its area is invaridated. In such case, use redraw() method which creates the exposed event.

```
object->clear();  
object->setAbsoluteDraw(True);  
object->draw();
```

- Clearing the instance (creates the exposed event)
You can redraw the instance with the method: `redraw()`. It creates the exposed event so that the other overlapped instances are updated. It can make the performance worse if there are many instances.

```
object->redraw();
```

10 How to move the position of the instances

You must clear the instances to move the position of it as follows.

```
void event_procedure(WSCbase* object){
    //clear the instance.
    object->clear();
    //set invisilbe.
    object->setVisible(False);
    //move the poosition.
    object->setProperty(WSNx,100);
    object->setProperty(WSNy,100);
    //set visible.
    object->setVisible(True);
}
```

There are some cases that an afterimage is left,if you move the position of the instace which has no window resource. Not to left if ,you have to clear the instance before move the position.

The instance which has a window resource is automaticary cleared.

11 How to create/delete the instances

11.1 How to create the instance

You can create the instance with the method: `WSCbase::getNewInstance()`.

```
char*    class_name = "WSCvlabel";
char*    obj_name   = "vlabel001";
WSCbase* parent    //The parent instance which has new instance.

//create a new instance.
WSCbase* object = WSCbase::getNewInstance(class_name,parent,obj_name);
object->initialize(); //initialize the instance.
object->clear();

object->setProperty(WSNx,100);
object->setProperty(WSNy,100);
object->setProperty(WSNwidth,100);
object->setProperty(WSNheight,100);
object->setVisible(True);
```

You have to call `initialize()` to initialize before calling the other methods of the created instance.

11.2 How to delete the instance

You can destroy the instance with the global function: `WSGFdestroyWindow()`.

```
//destroy the instance.
WSGFdestroyWindow(object);
```

Do not call `WSGFdestroyWindow()` with same instance twice, and do not access the destroyed instance, because it causes a fatal memory error.

12 How to use of the timer

You can execute the procedure after an interval or in cycles.

The timer class	Access function
WSDtimer	WSGIappTimer()

13 How to execute the procedure after an interval

At first, prepare the procedure to execute, and register it to the timer as trigger driven.

The method	Description
addTriggerProc()	adds procedures as trigger driven
delTriggerProc()	deletes a added procedures

```
#include "WSDtimer.h"
//the procedure which is executed by the timer ( trigger driven )
void triggerHandler(unsigned char clock,void* data){
    //The parameter: data is the third parameter of the method: addTriggerProc().

    //To do:
}

void event_procedure(WSCbase* obj){
    //this parameter is passed to the procedure.
    void* data = (void*)1234;
    //add the procedure to the timer (trigger driven) //after 1000ms
    long id = WSGIappTimer()->addTriggerProc( triggerHandler,WS1000MS,data );
    ...
    //if cancel...
    WSGIappTimer()->delTriggerProc( id );
}
```

You can implements the procedure which you want in "triggerHandler()", and pass some data by the third parameter: void* of WSDtimer::addTriggerProc(). WSDtimer::addTriggerProc() returns a timer id. you can cancel the timer by the id with WSDtimer::delTriggerProc().

Notice: After executing the procedure, it do not update the instances automatically, so you have to do it if needs.

```
//a sample of the trigger procedure.
void timerHandler(unsigned char clock,void* data){
    WSCbase* object = (WSCbase*)data;
    object->setProperty(WSNlabelString,"hello.");
    object->update(); //update the instance.
```

```
}

```

14 How to execute the procedure after in cycles

At first, prepare the procedure to execute, and register it to the timer as cycle driven.

The method	Description
addTimerProc()	adds procedures as cycle driven.
delTimerProc()	deletes added procedure.

```
#include "WSDtimer.h"
//the procedure which is executed by the timer ( cycle driven )
void timerHandler(unsigned char clock,void* data){
    //clock is a counter of the interval of 250ms
    //The parameter: data is the third parameter of the method: addTimerProc().

    //To do:
}

void event_procedure(WSCbase* obj){
    //this parameter is passed to the procedure.
    void* data = (void*)1234;
    //add the procedure to the timer (cycle driven) //500ms interval
    long id = WSGIappTimer()->addTimerProc( timerHandler,WS500MS,data );
    ..
    //if cancel..
    WSGIappTimer()->delTimerProc( id );
}

```

You can implements the procedure which you want in "timerHandler()", and pass some data by the third parameter: void* of WSDtimer::addTimerProc(). WSDtimer::addTimerProc() returns a timer id. you can cancel the timer by the id with WSDtimer::delTimerProc().

Notice: After executing the procedure, it do not update the instances automatically, so you have to do it if needs.

The cycles: WS250MS,WS500MS,WS750,WS1000MS,WS1250MS,... (250ms interval)

```
//a sample of the timer procedure.
void timerHandler(unsigned char clock,void* data){
    WSCbase* object = (WSCbase*)data;
    object->setProperty(WSNlabelString,"Hello!");
    object->update(); //update the instance
}

```

15 How to use the global key hook function

You can use the global key hook to check the keyboard event before dispatching, by adding a hook procedure to the keyboard instance.

The keyboard class	Access function
WSDkeyboard	WSGIappKeyboard()

```
#include <WSDkeyboard.h>
//A sample of the global key hook
WSCbool keyhandler(long keycode, Boolean onoff){
    // keycode : the key code (see WSkeysym.h)
    // onoff    : True = keypress, False= key release.
    if (keycode == WSK_F1){
        //The key code is F1...
        //if discard this...
        return False; //return the False: discard.
    }else if (keycode == WSK_F2){
        //The key code is F2...
        //if dispatch this...
        return True; //return the True: dispatch.
    }
    return True; //return the True: dispatch.
}
void event_procedure(WSCbase* obj){
    //the registration of the global key hook procedure.
    WSGIappKeyboard()->setGlobalKeyHook( keyhandler );
}
```

keyhandler() is the procedure which grabs the key board events to do something specially. You can register it with WSGIappKeyboard()->setGlobalKeyHook(),
See the header file: WSkeysym.h if you want the key codes.

16 How to select the key events on the input field

You can select or reject/convert the key events on the input field with the event procedure by the WSEV_KEY_HOOK trigger.

```
//A WSEV_KEY_HOOK sample procedure.
//Set the input field with the WSEV_KEY_HOOK trigger.
#include "WSDkeyboard.h"
...

void hookop(WSCbase* object){
    //get the pressed key code.
    long key = WSGIappKeyboard()->getKey();
    //Choose the key which is a numerical code.
    if ( (key >= WSK_0    && key <= WSK_9  ) ||
        (key >= WSK_KP_0 && key <= WSK_KP_9) ||
        key == WSK_plus    || key == WSK_minus ||
        key == WSK_BackSpace || key == WSK_Delete || key == WSK_Insert ||
        key == WSK_space   || key == WSK_Up    || key == WSK_Down    ||
        key == WSK_Left    || key == WSK_Right || key == WSK_Return ){
        //dispatches..
        return;
    }
    //Reject the other.
    WSGIappKeyboard()->setKey(0);
}
```

This sample of the input field shows how to choose the key event. WSGIappKeyboard()->setKey(0) reject the key event which you do not need.

17 How to add the event procedure on the programs

You can add the event procedure with the method: WSCbase::addProcedure().

The method	Description
addProcedure(WSCprocedure*)	Add the new event procedure.

Add the event procedure as follows. (A), Create the procedure instance with the specified procedure name and a trigger.

(B), Set a address of the procedure and its name with the method: setFunction(). (C) Finally, add the procedure instance to the instance.

```
void _new_event_procedure(WSCbase*){
//a sample of the event procedure.
//..
}

void event_procedure(WSCbase* object){
//Create the procedure instance which name is "new proc".
//The trigger is WSEV_MOUSE_IN (MOUSE_IN trigger.)
(A) WSCprocedure* ep = new WSCprocedure("new proc",WSEV_MOUSE_IN);
(B) ep->setFunction(_new_event_procedure,"_new_event_procedure");
(C) object->addProcedure(ep);
}
```

18 How to access to the arrayed instance.

It is possible to access to an arrayed instance as followings.

```
#include <WSCvlabel.h>

//extern declaration to access to an arrayed instance..
extern WSCvlabel** labelarray;

void event_procedure(WSCbase* object){
    labelarray[0]->setProperty(WSNlabelString,"Label No. 0");
    labelarray[1]->setProperty(WSNlabelString,"Label No. 1");
    ...
}
```

See Chapter 4.[How to define the instances as an array] of [Wide Studio Builder User's Guide] to create an arrayed instance.

19 How to indicate a dialog on the WSEV_EXIT event procedure

There is the case to need to save data indicate a dialog whether to finish the application, when the application is finished by closing the window. In such a case it is convenient to use the WSEV_EXIT event procedure of the WSCwindow/WSCmainWindow class.

The WSCwindow/WSCmainWindow class generates the WSEV_EXIT trigger when the window is disappeared before finishing the application.

At first set True to the property WSNexit of a WSCwindow/WSCmainWindow instance which is used as main window in the application, and put an event procedure to it with WSEV_EXIT trigger.

We will try to make an event procedure to have the following facility.

Indicates the dialog whether exit or continue.

If [OK] is selected,execute some procedure and exit the application.

If [NO] is selected,exit the application with no process.

If [CANCEL] is selected,do nothing and does not exit the application.

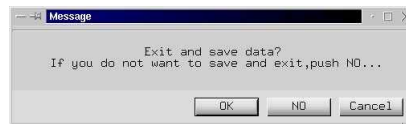
```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCmessageDialog.h>
#include <WSDtimer.h>

//The timer procedure which redisplay the window.
void delayproc(unsigned char,void* ptr){
    WSCbase* object = (WSCbase*)ptr;
    object->setVisible(True);
}
//EXIT event procedure
//Indicates a dialog.
void exit_ep(WSCbase* object){
    if (object->getVisible() != False){
        return;
    }
    WSCmessageDialog* msg = WSGIappMessageDialog(); //A
    msg->setProperty(WSNwidth,500);
    msg->setProperty(WSNno,True);
    msg->setProperty(WSNdefaultPosition,True);
    msg->setProperty(WSNlabelString,
        "Exit and save data?\n If you do not want to save and exit,push NO...");
    //Indicates the dialog.
    long ret = msg->popup(); //B
```

```
if (ret == WS_DIALOG_OK){ //When OK is selected.. C
    //saving some data ...
    exit(0);
}else
if (ret == WS_DIALOG_NO){ //When NO is selected.. D
    exit(0);
}else
if (ret == WS_DIALOG_CANCEL){ //When CANCEL is selected.. E
    WSGIappTimer()->addTriggerProc(delayproc,WS250MS,object);
}
}
static WSCfunctionRegister op("exit_ep", (void*)exit_ep);
```

Get the instance of the message dialog (A), indicate it (B). Check the result of the dialog (C)(D)(E).

It need to execute delayed procedure to redisplay the window, because it is required that the exit event is done before the window be redisplayed.



[The exit dialog]

20 How to examine which mouse button is pressed

There is a case to need to examine which the mouse button is pushed in some event procedure. It is possible to get a information of the mouse pointer from the global mouse instance as following program.

```
#include "WScom.h"
#include "WSCfunctionList.h"
#include "WSCbase.h"
//-----
//Function for the event procedure
//-----
#include "WSDmouse.h" //A

void btn_ep(WSCbase* object){
    long status = WSGIappMouse()->getStatus(); //B
    if (status & WS_MOUSE_BTN1){ //C
        //Left button is pushed..
    }
    if (status & WS_MOUSE_BTN2){ //D
        //Middle button is pushed..
    }
    if (status & WS_MOUSE_BTN3){ //E
        //Right button is pushed..
    }
}

static WSCfunctionRegister op("exit_ep", (void*)exit_ep);
```

At frust,include WSDmouse.h to access the global mouse instance (A). and, get a information of the mouse pointer (B). Check the button of the mouse whether to be pushed (C),(D),(E). It is better that cheking value by operator & than ==, because somethime the mouse buttons are pushed in same time.

B The sample of the event procedures

1 The sample of the event procedures for WSCvlabel

1.1 Making the WSCvlabel instance click-able

In the sample of the event procedures, It is a most basic procedure that makes the label instance to react to the mouse pointer. Here, you will create a procedure to make it counting up clicking of the mouse pointer.

```
#include "WSDmouse.h"
//Set this to a label instance with WSEV_MOUSE_PRESS(MOUSE-PRESS trigger)
void cbop(WSCbase* object){

    //(0) Which the mouse button is pressed?
    // btn1 -> fire. btn2 or the other -> return.
    if ( (WSGIappMouse()->getMouseStatus() & WS_MOUSE_BTN1) == 0){
        return;
    }

    //(A)Get the value of the property: WSNuserValue
    long value = object->getProperty(WSNuserValue);
    //(B)Count it up.
    value++;
    //(C)Display the value.
    object->setProperty(WSNlabelString,value);
    //(D)Store the counted value into the property: WSNuserValue for the next time.
    object->setProperty(WSNuserValue,value);
}
```

At first, this event procedure uses the property: WSNlabelString to display the number which is counted up. So, The kind of class like WSCvbtn,WSCvlabel which has it, can be used with WSEV_MOUSE_PRESS trigger. It will be executed by clicking of the mouse pointer.

- (0): It distinguishes whether the mouse btn 1 is pressed. Please refer to it for your implementation.
- (A): It uses the property: WSNuserValue to contain the value of the counter. The default value of the property is 0, and it can be used freely by user. The procedure uses it because it wants to store each counter value of each label instance. the counter value becomes a singleton when the procedure uses the static variable for the counter, even if used by many label instances.
- (B): It increases the counter.
- (C): It stores into the property: WSNlabelString to display it.

- (D): It stores into the property: WSNuseValue for the next time.

If by WSEV_MOUSE_IN trigger is used, it count the number of entranceing and exiting of the mouse pointer.

1.2 Making the WSCvlabel instance select-able

Here, you will create a procedure to make the label instance select-able by clicking of the mouse pointer. To display the instance is selected, the procedure changes the bgcolor of it. This time, the procedure uses the method: set/getUserData() to get/store data instead of the property: WSNuserValue.

```
//Set this to a label instance with WSEV_MOUSE_PRESS(MOUSE-PRESS trigger)
void cbop(WSCbase* object){
    //(A) Get the value with getUserData()
    long value = (long)object->getUserData("STATUS");
    //(B) it makes the instance selected if value is 0, and unselected if 1.
    if (value == 0){
        //(C) Store the bgcolor(which is string type) into WSNuserString
        WSCvariant color = object->getProperty(WSNbackColor);
        object->setProperty(WSNuserString,color);
        //(D)Set the bgcolor to the selected color.
        object->setProperty(WSNbackColor,"slategray4");
        //(E)Store the state with setUserData().
        value = 1;
        object->setUserData("STATUS", (void*)value);
    }else{
        //(F)Get the original bgcolor from WSNuserString.
        WSCvariant color = object->getProperty(WSNuserString);
        //(G)Store it to WSNbackColor to display with the original color.
        object->setProperty(WSNbackColor,color);
        //(H)Store the state with setUserData().
        value = 0;
        object->setUserData("STATUS", (void*)value);
    }
}
```

The kind of class like WSCvbtn,WSCvlabel which has the property: WSNbackColor, can be used with WSEV_MOUSE_PRESS trigger. It will be executed by clicking of the mouse pointer.

- (A): It uses the method: set/getUserData() to contain the selected value of the state. The default value of the method is 0, and it can be used freely by user. The procedure uses it because it wants to store each status of each instance.

You can specify a name of value to store with `setUserData()`, and can get the value by the specified name with `getUserData()`.

- (B): It distinguishes the state.
- (C): It stores the original bgcolor to the property: `WSNuserString`.
- (D): It makes the instance selected.
- (E): It stores the state with `setUserData()` again.
- (F): It gets the original bgcolor from `WSNuserString`.
- (G): It stores the original bgcolor to `WSNBackColor`.
- (H): It stores the state with `setUserData()` again.

1.3 Making the WSCvlabel instance highlight-able

Here, you will create a procedure with `WSEV_MOUSE_IN/OUT` to make the instance highlighted. Coming into the area, the instance is highlighted, and going out of the area, it is returned normal.

An important matter is that you create a procedure which prepares a sub-procedure with `WSEV_MOUSE_IN` and another with `WSEV_MOUSE_OUT`. In other words, that procedure with `WSEV_INITIALIZE` is executed, it adds two sub-procedures to the instance which trigger is `WSEV_MOUSE_IN` and `WSEV_MOUSE_OUT` to make the instance highlight-able. One procedure can prepare many procedures. Then you can go with a procedure even if many procedures are needed.

```
//a sub-procedure with WSEV_MOUSE_IN trigger
void subop1(WSCbase* object){
    //(A)Store the original bgcolor to WSNuserString
    WSCvariant color = object->getProperty(WSNBackColor);
    object->setProperty(WSNuserString,color);
    //(B)highlight the instance.
    object->setProperty(WSNBackColor,"slategray4");
}
//a sub-procedure with WSEV_MOUSE_OUT trigger
void subop1(WSCbase* object){
    //(C)Get the original bgcolor from WSNuserString
    WSCvariant color = object->getProperty(WSNuserString);
    //(D)Store the original bgcolor.
    object->setProperty(WSNBackColor,color);
}
//a main-procedure with WSEV_INITIALIZE trigger
void cbop(WSCbase* object){
    //If executed,it add the sub-procedures to the instance.
    //(E) Setup a sub-procedure:WSEV_MOUSE_IN.
    //ProcedureName="Highlight1"      Trigger=WSEV_MOUSE_IN      Function=subop1
    WSCprocedure* ac1 = new WSCprocedure("Highlight1",WSEV_MOUSE_IN);
    ac1->setFunction(subop1,"subop1");
    object->addProcedure(ac1);
}
```



```

// (F) Setup a sub-procedure: WSEV_MOUSE_OUT.
// ProcedureName="Highlight2"      Trigger=WSEV_MOUSE_OUT      Function=subop2
WSCprocedure* ac2 = new WSCprocedure("Highlight2", WSEV_MOUSE_OUT);
ac2->setFunction(subop2, "subop2");
object->addProcedure(ac2);
}

```

The subop1() is executed by WSEV_MOUSE_IN fired, and makes the instance bgcolor highlight(A)(B). The subop2() is executed by WSEV_MOUSE_OUT fired, and makes the instance bgcolor original one(C)(D). The main procedure is executed by WSEV_INITIALIZE only once to setup the sub-procedures (E)(F).

1.4 Making a group of selectable WSCvlabel instances

Here, you create an event procedure to make a group of the mouse-selectable label instances on the same parent. The procedure makes the instance selected by storing WS_SHAODW_IN to the property: WSNshadowType and memorises which instance is selected by storing it to its parent instance.

```

// An event procedure with WSEV_MOUSE_PRESS trigger
void cbop(WSCbase* object){
    // (A) Use the value of WSNuserValue as "instance identifier"
    long val = object->getProperty(WSNuserValue);
    // (B) Make the instance selected: WS_SHADOW_IN state.
    object->setProperty(WSNshadowType, WS_SHADOW_IN);
    // (C) Get the last selected instance which is memorized
        with setUserData() of the parent instance.
    WSCbase* parent = object->getParent();
    WSCbase* target = (WSCbase*)parent->getUserData("SelectedItem");
    // (D) Make it not selected: WS_SHADOW_OUT state.
    if (target != NULL){
        target->setProperty(WSNshadowType, WS_SHADOW_OUT);
    }

    if (target == object){
        // (E) When clicking the selected instance twice,
        // clear the selected state.
        parent->setUserData("GroupValue", (void*)0);
        parent->setUserData("SelectedItem", (void*)0);
    }else{
        // (E) The other, store the selected instance to the parent instance.
        parent->setUserData("GroupValue", (void*)val); // Instance identifier
        parent->setUserData("SelectedItem", (void*)object); // selected instance
    }
}
}

```

The label instances need each instance identifier to recognise which instance is selected, then we decide to use the property: WSNuserValue as the instance identifier which has a unique value.

- (A): Gets the instance identifier from the property: WSNuseValue.
- (B): Makes the instance selected with WS_SHAODW_IN.
- (C): Gets the last instance which is selected from the parent instance.
- (D): Makes the last one not selected.
- (E): Makes the instance not selected if it is selected twice and clears the value which is memorized in the parent instance.
- (F): Stores the new selected instance to the parent instance.

2 The sample of the event procedures for WSCvifield

2.1 Executing some event procedures by return key

You can execute the specified procedures in the event procedure. Here, the following sample shows executing the procedure which name is "InputFixed".

```
//A sample of WSEV_KEY_HOOK trigger.
#include "WSDkeyboard.h"
void cbop(WSCbase* object){
    (A)Get the pressed key.
    long key = WSGIappKeyboard()->getKey();
    (B)If the key is return..
    if (key == WSK_Return){
        //Execute the procedure which name is "InputFixed".
        object->execProcedure("InputFixed");
    }
}
```

- (A): Get the key code which is pressed from the keyboard.
- (B): if it is return key, do (C).
- (C): Execute the procedure which name is "InputFixed".

This procedure sometimes is used to execute some procedure on the end of key input.

2.2 Clearing the last input string on starting of next input

Here, you create a procedure which clears the last input string on starting of the next. the procedure clears the string the first key input since the input field instance is focused or clicked by the mouse pointer. The clear process is as follows:

- (1)It sets the clear flag True,if the input field is focused.
- (2)It sets the clear flag True,if the input field is clicked.
- (3)It clears the string if the clear flag is True.
- (4)It initializes the sub-procedures (1),(2),(3).

```
#include <WSDkeyboard.h>
//To contain the last focused input field.
static WSCbase* _focus_if = NULL;
//(1)A sub-procedure with WSEV_FOCUS_CH
static void _focus_on_(WSCbase* object){
    //(A) Examine whether the instance is focused.
```

```

    if (_focus_if != object && object->getFocus() != False){
        //(B)It need to clear the string!
        //Set the clear flag True.
        object->setUserData("CLEAR TIMING",(void*)1);
        //(C)store that the last focused one is.
        _focus_if = object;
    }
}
//(2)A sub-procedure with WSEV_MOUSE_PRESS
static void _btn_press_(WSCbase* object){
    //(D) if clicked by the mouse pointer,
    // it needs to clear the string!
    // Set the clear flag True.
    object->setUserData("CLEAR TIMING",(void*)1);
    object->setProperty(WSNcursorPos,0);
    //(E)store that the last focused one is.
    _focus_if = object;
}
//(3)A sub-procedure with WSEV_KEY_PRESS
static void _key_hook_(WSCbase* object){
    //(F) See the clear flag to clear the last input string.
    long fl =(long)object->getUserData("CLEAR TIMING");
    if (fl == 1){
        long key = WSGIappKeyboard()->getKey();
        //(G) Clear the string,if the key is not cursor key.
        if (key != WSK_Tab &&
            key != WSK_Up &&
            key != WSK_Down &&
            key != WSK_Left &&
            key != WSK_Right ){
            //(H)Clear...
            object->setProperty(WSNlabelString,"");
        }else{
            return;
        }
    }
    //(I)Set the clear flag False.
    object->setUserData("CLEAR TIMING",(void*)0);
}

//The main-procedure.
//(4)Set the input field instance with WSEV_INITIALIZE trigger.
void ifdclr(WSCbase* object){
    //Setup the sub-procedure(1) with WSEV_FOCUS_CH
    WSCprocedure* ac1 = new WSCprocedure("ac1",WSEV_FOCUS_CH);
    ac1->setFunction(_focus_ch_,"_focus_ch_");
    object->addProcedure(ac1);
}

```

```

//Setup the sub-procedure(2) with WSEV_MOUSE_PRESS
WSCprocedure* ac2 = new WSCprocedure("ac2",WSEV_MOUSE_PRESS);
ac2->setFunction(_btn_press_, "_btn_press_");
object->addProcedure(ac2);

//Setup the sub-procedure(3) with WSEV_KEY_PRESS
WSCprocedure* ac3 = new WSCprocedure("ac3",WSEV_KEY_HOOK);
ac3->setFunction(_key_hook_, "_key_hook_");
object->addProcedure(ac3);
}

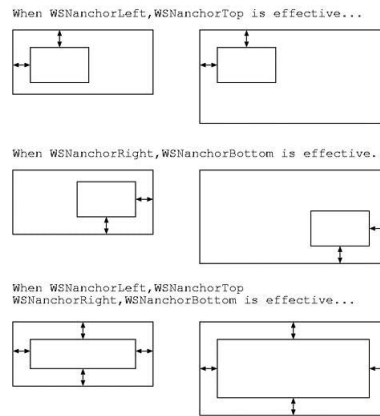
```

In the focus_ch event procedure, To examine the instance is focused afresh, It uses the static variable which is stored the last focused instance.

- (A): It checks whether the instance is equal to the last focused one. if it differs and it focused (not losed), it means that the instance is focused afresh.
- (B): It sets the clear flag True.
- (C): It stores the instance to the static variable for the next.
- (D): It sets the clear flag True and move the cursor to the top.
- (E): It stores the instance to the static variable for the next.
- (F): It sees the clear flag to clear the last string.
- (G): It is too sad to clear the string with the cursor key, so it sees what it is.
- (H): It clears the string if it is not.
- (I): It sets the clear flag False.

3 Automatic geometry adjustment with the anchors

The object of WideStudio which has geometry has automatic geometry adjustment by the property WSNAnchor. The anchor keeps the distance of the border of parent instance and coordinates the geometry of oneself.



[Action of the anchors]

4 The pulldown menu and the menu area

4.1 What is the menu area

The menu area: WSCmenuArea is a form which secures an area for the pulldown menus. This class keeps constant area the upper part. And its geometry is automatically adjusted when the parent window is resized.



```

File
-----
// Wide Studio Application C++ Source File
// created by Wide Studio source generator
#include <WSCom.h>
#include <WSCappDev.h>

//--- OBJECT includes ---//
#include <WSCmainWindow.h>
#include <WSCmenuArea.h>
#include <WSCpulldownMenu.h>
#include <WSCtextField.h>

//--- OBJECT instance variable ---//
WSCmainWindow* newwin000;
WSCmenuArea* newmenu_000;
WSPulldownMenu* newpulld_001;
WSCtextField* newtext_002;

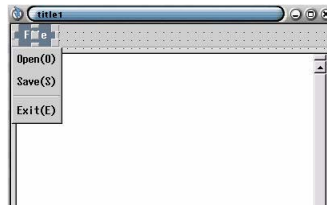
//--- OBJECT src ---//
void _create_win_newwin000()

```

[A sample which has a menu area(WSCmenuArea)]

4.2 Try to use the pulldown menu

Now try to create a sample application which has pulldown menu.



[A sample application which has pulldown menu]

The sample has [File] menu such as picture and the [File] menu has these items as follows.

- Open
- Save
- Exit

At first, create an application window and drop an instance of WSCmenuArea from the section [Forms] of object box dialog. Next, drop an instance of WSCpulldownMenu from the section [Commands] to it. Set the property as follows.

Label string: File

Menu items: Open(O):open_ep:o,Save(S):save_ep:s,SP,Exit(E):exit_ep:e

To the menu items property,set value which is punctuated with comma every one item.

One item consists of the following format. It indicates a separator with an item: SP.

```
item string:event procedure name:short cut key,...
```

For example, in an above-mentioned example, an event procedure which title is open_ep is executed when the menu [Open(0)] selected. You can set event procedures which title is open_ep with NONE trigger to the pulldown menu instance.

You can assign ID to each menu item, and you can process the all triggers of each menu with an event procedure to use the ID.

In the case with ID, one item consists of the following format. In the procedure,you can get the ID with getValue() method.

```
item string:event procedure name:short cut key:id,...
```

4.3 Notice of the pulldown menu

It is no allowed that using WSCpulldownMenu class and other WSCvXXXX class on a same form. Place instances of WSCpulldownMenu class only on a menu area.

5 The list

5.1 Setting of the items by the method

Use addItem method to add items to the list with an string and position.

If the position is omitted,add it to the last of the list. The following program is a sample adding item to the list.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSClist.h>
extern WSClist* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list.
    newlist_001->delAll();
    //Add an item to the last of the list.
    newlist_001->addItem("item1");
    newlist_001->addItem("item2");
    newlist_001->addItem("item3");
    newlist_001->addItem("item4");
    //Add an item to the specified position of the list.
    newlist_001->addItem("item0",0);//0 :top
    newlist_001->addItem("item5",-1);//-1 :last

    //Update the modified list.
    newlist_001->updateList();
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);
```

5.2 Setting of the items by the property

the property WSNdata can be used for setting the items of the list, if it is a comparatively little number of the items. In such case, set the property WSNdataSource to WS_DATA_SOURCE_NONE, and set the items which is punctuated with comma every one item to the proeprty WSNdata.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSClist.h>
extern WSClist* newlist_001;
void btnep1(WSCbase* object){
```

```

//Delete all the items of the list.
newlist_001->delAll();
//Set the items of the list by the property
newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_NONE);
newlist_001->setProperty(WSNdata,"item0\nitem1\nitem2\nitem3\nitem4");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);

```

It is possible to indicate the specified ICON by icon file name as following program. Set True to the property WSNuseIcon and Set the items as following format. If it is omitted, the value of WSNiconPixmap is used for default icon.

```

format:
    the ICON1 filename,the string2 of the item\n the ICON2 filename,the string2 of the item\n...

void btnep1(WSCbase* object){
//Delete all the list of the list
newlist_001->delAll();
//Set the items of the list by the property
newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_NONE);
newlist_001->setProperty(WSNdata,
    "$(WSDIR)/sys/pixmaps/bi16.xpm,item1\nitem2\nitem3");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);

```

5.3 Setting of the items from the file directly

It is possible to set the items from the file directly. In such case at first, set WS_DATA_SOURCE_FILE to the property WSNdataSource, and set the file name to the property WSNdataSourceName as following program.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSClist.h>
extern WSClist* newlist_001;
void btnep1(WSCbase* object){
//Delete all the items of the list
newlist_001->delAll();
//Set the items from file directly
newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_FILE);
newlist_001->setProperty(WSNdataSourceName,"data.txt");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);

```

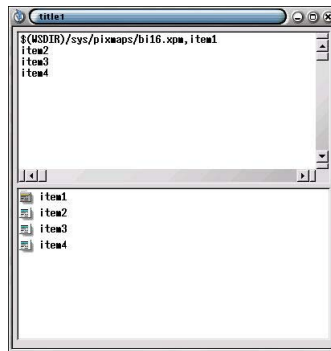
```
//data.txt contains...
$(WSDIR)/sys/pixmaps/bi16.xpm,item1
item2
item3
item4
$(WSDIR)/sys/pixmaps/bi16.xpm,item5
item6
item7
item8
```

5.4 Setting of the items from the other instance

It is possible to set the items from the data source target property of the other instances. In such case, set `WS_DATA_SOURCE_INSTANCE` to the property `WSNdataSource`, and set the instance name to the property `WSNdataSourceName`. In the following example, a `WSCtextField` instance are specified to the property `WSNdataSourceName` and then it sets the string data of it to the list.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSClist.h>
extern WSClist* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list
    newlist_001->delAll();
    //Set the items from the other instances
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_INSTANCE);
    newlist_001->setProperty(WSNdataSourceName,"newtext_000");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);
```

The following picture indicates the list(lower side) from the other instance of `WSCtextField` class(upper side).

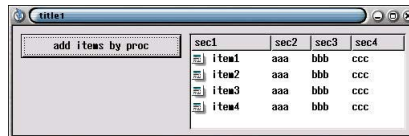


[The list data from the other instance]

6 The verbose list

6.1 Setting of the items by the method

Use addItem method to add items to the list which property WSNTtype is WS_VERBOSE with an string and position. If the position is omitted,add it to the last of the list. The following program is a sample adding item to the list. The difference from the list is the item which is punctuated with comma.



[The verbose list]

Set 50,100,150 to the property WSNbarValue to make the list 4-sections like above picture. This property appoints a position of separator of title. Please pay attention, because number of section is fixed by this property.

The following program is a sample adding item to the list.

```
#include <WSCom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCverbList.h>
extern WSCverbList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list.
    newlist_001->delAll();
    //Add an item to the last of the list.
    newlist_001->addItem("item1,aaa,bbb,ccc");
    newlist_001->addItem("item2,aaa,bbb,ccc");
    newlist_001->addItem("item3,aaa,bbb,ccc");
    newlist_001->addItem("item4,aaa,bbb,ccc");

    //Add an item to the specified position of the list.
    newlist_001->updateList();
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);
```

6.2 Setting of the items by the property

the property WSNdata can be used for setting the items of the list, if it is a comparatively little number of the items. In such case, set the property WSNdataSource to WS_DATA_SOURCE_NONE, and set the items which is punctuated with comma every one item to the proeprty WSNdata.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCverbList.h>
extern WSCverbList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list.
    newlist_001->delAll();
    //Set the items of the list by the property
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_NONE);
    newlist_001->setProeprty(WSNdata,
        "item1,aaa,bbb,ccc\nitem2,aaa,bbb,ccc\nitem3,aaa,bbb,ccc");

static WSCfunctionRegister  op("btnep1",(void*)btnep1);

```

It is possible to indicate the specified ICON by icon file name as following program. Set True to the proeprty WSNuseIcon and Set the items as following format. If it is ommited,the value of WSNiconPixmap is used for default icon.

```

format:
    the ICON1 filename,str11,str12,str13,.. \nthe ICON2 filename,str21,str22,str23,... \n...

void btnep1(WSCbase* object){
    //Delete all the list of the list
    newlist_001->delAll();
    //Set the items of the list by the property
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_NONE);
    newlist_001->setProperty(WSNdata,
        "$ (WSDIR)/sys/pixmaps/bi16.xpm,item1,aaa,bbb,ccc\nitem2,aaa,bbb,ccc\nitem3,aaa,bbb,ccc");
}
static WSCfunctionRegister  op("btnep1",(void*)btnep1);

```

6.3 Setting of the items from the file directly

It is possible to set the items from the file directly. In such case at first,set WS_DATA_SOURCE_FILE to the property WSNdataSource, and set the file name to the property WSNdataSourceName as following program.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCverbList.h>

```

```

extern WSCverbList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list
    newlist_001->delAll();
    //Set the items from file directly
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_FILE);
    newlist_001->setProperty(WSNdataSourceName,"data.txt");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);

//data.txt contains...
$(WSDIR)/sys/pixmaps/bi16.xpm,item1,aaa,bbb,ccc
item2,aaa,bbb,ccc
item3,aaa,bbb,ccc
item4,aaa,bbb,ccc
$(WSDIR)/sys/pixmaps/bi16.xpm,item5,aaa,bbb,ccc
item6,aaa,bbb,ccc
item7,aaa,bbb,ccc
item8,aaa,bbb,ccc

```

6.4 Setting of the items from the other instance

It is possible to set the items from the data source target property of the other instances. In such case, set `WS_DATA_SOURCE_INSTANCE` to the property `WSNdataSource`, and set the instance name to the property `WSNdataSourceName`. In the following example, a `WSCtextField` instance are specified to the property `WSNdataSourceName` and then it sets the string data of it to the list.

```

#include <WSCom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCverbList.h>
extern WSCverbList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list
    newlist_001->delAll();
    //Set the items from the other instances
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_INSTANCE);
    newlist_001->setProperty(WSNdataSourceName,"newtext_000");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);

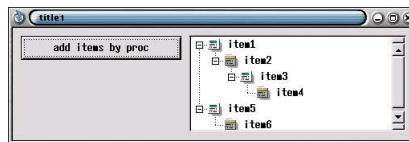
```

7 The tree list

7.1 Setting of the items by the method

Use addItem method to add items to the list which property WSNtype is WS_TREE with a string and position. If the position is omitted, add it to the last of the list. The following program is a sample adding item to the list. The difference from the list is the item which is punctuated with comma. In addition, it is possible to set the nest floor of list item with the setItemValue method. In its argument, set the position of the item and WS_INDENT_LEVEL to the value type, and indent level. If it is omitted, the indent level is 0.

```
setItemValue(pos,WS_INDENT_LEVEL,level);
pos = 0,1,2,...,-1(last one)
level = 0(top),1,2,3...
```



[The tree list]

The following program is a sample adding item to the list.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCtreeList.h>
extern WSCtreeList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list.
    newlist_001->delAll();
    //Add an item to the last of the list.
    newlist_001->addItem("item1");
    newlist_001->setItemValue(-1,WS_INDENT_LEVEL,0);
    newlist_001->addItem("item2");
    newlist_001->setItemValue(-1,WS_INDENT_LEVEL,1);
    newlist_001->addItem("item3");
    newlist_001->setItemValue(-1,WS_INDENT_LEVEL,2);
    newlist_001->addItem("item4");
    newlist_001->setItemValue(-1,WS_INDENT_LEVEL,3);
    newlist_001->addItem("item5");
    newlist_001->setItemValue(-1,WS_INDENT_LEVEL,0);
    newlist_001->addItem("item6");
```



```

newlist_001->setItemValue(-1,WS_INDENT_LEVEL,1);

//Add an item to the specified position of the list.
newlist_001->updateList();
}
static WSCfunctionRegister op("btnep1", (void*)btnep1);

```

As for the point that you should pay attention to in tree list, there is not special relational as membership between item at all. and then each item is merely indicated by status appointed indent. So by deleting the item of the upper level, the item of floor following it is not disappeared. The difference of upper level is only +1. If it is grater than +1, it is adjusted to +1 automatically.

7.2 Setting of the items by the property

the property WSNdata can be used for setting the items of the list, if it is a comparatively little number of the items. In such case, set the property WSNdataSource to WS_DATA_SOURCE_NONE, and set the items which is punctuated with comma every one item to the proeprty WSNdata.

```

Format:(the property WSNuseIcon is True)
  icon_filename,indent_level,1=open/0=close,the string\n...
Format:(the property WSNuseIcon is False)
  indent_level,1=open/0=close,the string\n...

```

If the icon is ommited in the case that the property WSNuseIcon is True, the value of WSNiconPixmap is used for the icon.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCtreeList.h>
extern WSCtreeList* newlist_001;
void btnep1(WSCbase* object){
  //Delete all the items of the list.
  newlist_001->delAll();
  //Set the items of the list by the property
  newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_NONE);
  newlist_001->setProeprty(WSNdata,"0,1,item1\n,1,1,item2\n,2,1,item3");

static WSCfunctionRegister op("btnep1", (void*)btnep1);

```

7.3 Setting of the items from the file directly

It is possible to set the items from the file directly. In such case at first, set WS_DATA_SOURCE_FILE to the property WSNdataSource, and set the file name to the property WSNdataSourceName as following program.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCtreeList.h>
extern WSCtreeList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list
    newlist_001->delAll();
    //Set the items from file directly
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_FILE);
    newlist_001->setProperty(WSNdataSourceName,"data.txt");
}
static WSCfunctionRegister op("btnep1",(void*)btnep1);

//data.txt contains...
$(WSDIR)/sys/pixmaps/bi16.xpm,0,1,item1
1,1,item2
2,1,item3
3,1,item4
$(WSDIR)/sys/pixmaps/bi16.xpm,0,1,item5
1,1,item6
2,1,item7
3,1,item8

```

7.4 Setting of the items from the other instance

It is possible to set the items from the data source target property of the other instances. In such case, set `WS_DATA_SOURCE_INSTANCE` to the property `WSNdataSource`, and set the instance name to the property `WSNdataSourceName`. In the following example, a `WSCtextField` instance are specified to the property `WSNdataSourceName` and then it sets the string data of it to the list.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCtreeList.h>
extern WSCtreeList* newlist_001;
void btnep1(WSCbase* object){
    //Delete all the items of the list
    newlist_001->delAll();
    //Set the items from the other instances
    newlist_001->setProperty(WSNdataSource,WS_DATA_SOURCE_INSTANCE);
}

```

```
    newlist_001->setProperty(WSNdataSourceName,"newtext_000");  
}  
static WSCfunctionRegister  op("btnep1",(void*)btnep1);
```

8 The user dialog

8.1 How to make a simple user dialog

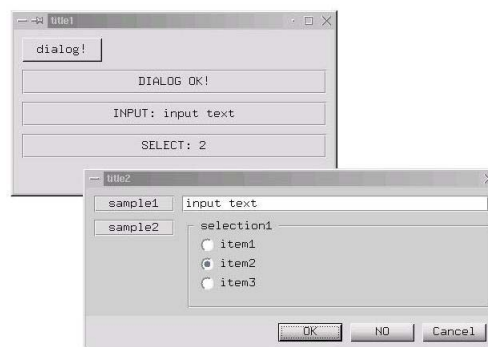
Try to make a simple user dialog with WSCdialog class. and the application has as following functions.

Pushing some button of the application,indicates this user dialog.

This dialog has an input field and a radio group.

The end of input of the dialog,check the input value whether it is right.

Indicates the result of input of the dialog on the label.



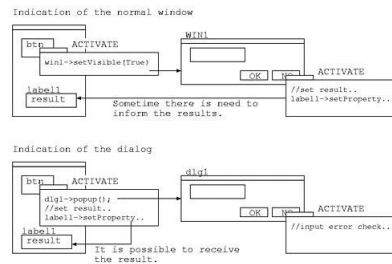
[A sample of the user dialog]

This sample is provided by `ws/samples/share/dialog/newproject.jpg`.

8.2 Controlling to indicate the user dialog

It is easy to see to control indicating the user dialog by a sample `ws/sampes/share/dialog/newproject.prj`. The dialog is often used as exclusive window to do the input and indicating some informations. So there is a bad case when it is implemented as usual window. For example,when it requires that the same dialog is called from more than on event procedure,if the dialog is usual window, it becomes complicated that receiving of the input value from it. But it is easy to receive it when it is as dialog,not as usual window, because the popup method of dialog returns the end of input in order to receive the input value form it.

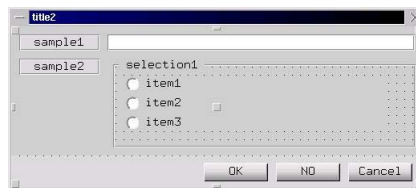
The following picutre shows the difference of the indicating and receiving the input data between the usual window and the dialog.



[The difference of the usual window and the dialog]

See the following program, At first, make a user dialog instance by dropping from the [Window] section of the object box dialog and put the following instances on it.

```
WScvifield* newvifi_003
WScradioGroup* newradi_006
```



[A sample of the user dialog]

The next,implement the procedure when the button [OK],[NO],[CANCEL] is pushed, and put it the dialog with WSEV_ACTIVATE trigger. In this procedure, check the input values whether they are right and indicates an error dialog if they are wrong.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCdialog.h>
#include <WScvifield.h>
#include <WScradioGroup.h>
#include <WSCmessageDialog.h>
extern WScvifield* newvifi_003;
extern WScradioGroup* newradi_006;

void dialogep(WSCbase* object){
    WSCdialog* dialog = (WSCdialog*)object->cast("WSCdialog");
    if (dialog == NULL){ //A
```

```

    return;
}
if (dialog->getStatus() != WS_DIALOG_OK){ //B
    object->setVisible(False);
    return;
}

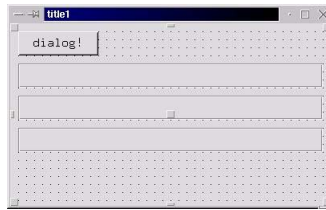
WSCstring str;
str = newwifi_003->getProperty(WSNlabelString);
if (!strcmp((char*)str,"")){ //C
    WSCmessageDialog* msg = WSGIappMessageDialog();
    msg->setProperty(WSNdefaultPosition,True);
    msg->setProperty(WSNwidth,500);
    msg->setProperty(WSNlabelString,"Please input some string to the input field.");
    msg->popup(); //D
    return;
}
long val = newradi_006->getProperty(WSNvalue);
if (val == 0){ //E
    WSCmessageDialog* msg = WSGIappMessageDialog();
    msg->setProperty(WSNdefaultPosition,True);
    msg->setProperty(WSNwidth,500);
    msg->setProperty(WSNlabelString,"Please select a item of the radio group.");
    msg->popup(); //F
    return;
}
object->setVisible(False); //E
}
static WSCfunctionRegister op("dialogep",(void*)dialogep);

```

At first, get the class native pointer to access the original method of the WSCdialog class. and call the getStatus method to receive an information which button [OK],[NO],[CANCEL] is pushed. At A,exits the procedure if the instance is not WSCdialog class. At B,checks which the button is pushed. Puts the dialog out when the button is not [OK]. At C, checks the input of the instance: newwifi_003, and indicates an message dialog when its input is wrong at D and exits, At C, checks the input of the instance: newradi_006, and indicates an message dialog when its input is wrong at F and exits.

Then puts the dialog out and the method popup() which is called to indicate this dialog and called this event procedure by pushing the buttons of dialog returns. It is important that to make the dialog disappeared, because if not,the method popup() will never return.

The following program is an example to call the method popup() in order to indicate the dialog.



[A sample window to indicate the user dialog]

If the button [dialog!] is pushed, indicates the user dialog and receive the input values from the dialog, put them to the labels.

The first label: The pushed button [OK],NO],[CANCEL].

The second label: The input of newvifi_003.

The third label: The selection of newradi_006.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCdialog.h>
#include <WSCvifield.h>
#include <WSCradioGroup.h>
#include <WSCvlabel.h>
extern WSCdialog* newdial_001;
extern WSCvifield* newvifi_003;
extern WSCradioGroup* newradi_006;
extern WSCvlabel* newvlab_007;
extern WSCvlabel* newvlab_010;
extern WSCvlabel* newvlab_011;

void btnep(WSCbase* object){
    long val = newdial_001->popup();
    if (val == WS_DIALOG_OK){
        newvlab_007->setProperty(WSNlabelString,"DIALOG OK!");
    }else
    if (val == WS_DIALOG_NO){
        newvlab_007->setProperty(WSNlabelString,"DIALOG NO!");
    }else
    if (val == WS_DIALOG_CANCEL){
        newvlab_007->setProperty(WSNlabelString,"DIALOG CANCEL!");
    }
    WSCstring tmp;
    tmp = newvifi_003->getProperty(WSNlabelString);
    WSCstring tmp2;
    tmp2 << "INPUT: " << tmp;
}
```

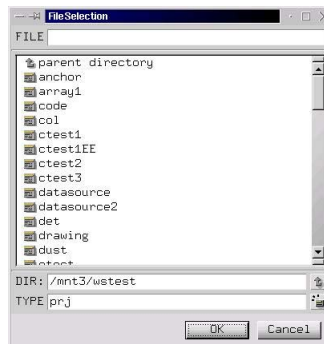
```
newvlab_010->setProperty(WSNlabelString,tmp2);

val = newradi_006->getProperty(WSNvalue);
tmp2 = "SELECT: ";
tmp2 << val;
newvlab_011->setProperty(WSNlabelString,tmp2);
}
static WSCfunctionRegister op("btnep",(void*)btnep);
```


9 The file selection dialog

9.1 Indication of the file selection dialog

To get the instance of the file selection dialog, call the global function: `WSGIappFileSelection()`. and call the dialog method: `popup()` to indicate it.



[The file selection dialog]

The method `popup` returns the result of the dialog status when the selection is done. And get the selected file name by the `getFileName()` method.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include <WSCfileSelect.h> //(A)
#include <WSCmessageDialog.h> //(B)
//-----
//Function for the event procedure
//-----
void btnep2(WSCbase* object){
    //Access to the global instance of the WSCfileSelection class.
    WSCfileSelect* fs = WSGIappFileSelect(); //(C)
    fs->setProperty(WSNmaskFileName,"cpp"); //(D)
    fs->setProperty(WSNdefaultPosition,True); //(E)
    long ret = fs->popup(); //(F)

    //Access to the global instance of the WSCmessageDialog class.
    WSCmessageDialog* msg = WSGIappMessageDialog(); //(G)
    msg->setProperty(WSNwidth,500); //(H)
    msg->setProperty(WSNheight,120); //(I)
    msg->setProperty(WSNdefaultPosition,True); //(J)

    if (ret == WS_DIALOG_OK){ //(K)
        WSCstring str;
        str << fs->getFileName() << " is selected.";
```

```

    msg->setProperty(WSNlabelString,str);
    msg->popup();
}else if (ret == WS_DIALOG_NO){    //(L)
    msg->setProperty(WSNlabelString,"Nothing is selected.");
    msg->popup();
}else if (ret == WS_DIALOG_CANCEL){
    msg->setProperty(WSNlabelString,"The selection is canceled.");
    msg->popup();                //(M)
}
}
}

```

At first, include the header,WSCfileSelect.h, access to them. A global instance of WSCfileSelect class is already prepared, and it is possible to access by the global function WSGIappFileSelect() at (C). Set the property of it at (D),(E), and indicate it by the popup() method at (F), and then the popup method returns that selection is done.

To see the result the file selection dialog,try to indicates the result by the message dialog. At (G),(H),(I),(J),get the global instance of the message dialog and set its property up. At (K),indicate the selected file name. At (L),indicate [Nothing is selected]. At (M),indicate [The selection is canceled]

10 The scrolled form

10.1 How to use the virtual scrolling

The large scrolling area uses huge window resource. In such case, it requires to use the virtual scrolling which function uses no window resource.

The virtual scrolling has the following function.

it requires no window resource for any scrolling size.

The virtual scrolling has the following weak points.

Drawing performance is worse than normal scrolling mode.

The instances which have a window resource can not be placed on the virtual scrolling mode.

The does not move by scrolling the area.

It is possible to use the instances which have no window resource and which class name is WSCvxxxx on the virtual scrolling.

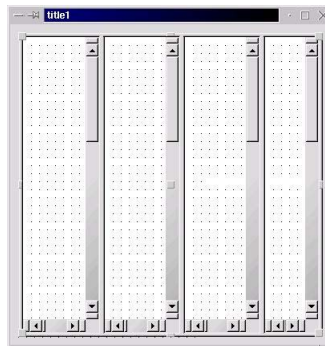
11 The separated form

The separated form has some area separated by some separators which can be move by the mouse pointer.

12 How to set the width of the separated area

At first, decide a direction to divide horizontal and vertical, set it to the property WSNorientation. Next, set the position the separators to the property WSNbarValue.

For example,if the direction is vertical,and 3 separators are available,in other words,4 areas are available, to set the position of the separators 100,200,300 dots from left side, specify the property WSNbarValue 100,200,300.



[The separated form with 3 separators which position is 100,200,300 which contains the scrolled form on each area]

13 The drawing area

13.1 How to draw pictures on the drawing area

It is possible to draw pictures freely by the drawing area. It has the methods to draw various pictures which can be used on the event procedure with exposure event WSEV_EXPOSE.

The following program shows a basic method to draw pictures by the drawing area.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdrawingArea.h>
#include <WSCvslider.h>

void drawep(WSCbase* object){
    //drawing_a is same as newvdra_000...
    //You can get it extern WSCvdrawingArea* newvdra000; also.
    WSCvdrawingArea* drawing_a =
        (WSCvdrawingArea*)object->cast("WSCvdrawingArea"); //(A)
    if (drawing_a == NULL){ //(B)
        return;
    }

    drawing_a->setForeColor("#ff0000"); //(C)
    drawing_a->drawLine(0,0,100,100); //(D)
}

static WSCfunctionRegister op("drawep", (void*)drawep);
```

At first access to the method of the drawing area, include the header WSCdrawingArea.h of the WSCdrawingArea class and get the native class pointer of the WSCdrawingArea class at (A). It is impossible to access the native method of WSCdrawingArea class with the pointer of WSCbase class. If the pointer drawing_a is NULL at (B), the instance is not WSCdrawingArea, so exit the event procedure. Next, it is the sample to set the color to the drawing area which is used to the other methods for drawing pictures (C). At (D), draws the line from (0,0) to (100,100).

The drawing area class has the following methods.

- The rectangle and the filled rectangle
- The circle, filled circle, arc, chord, oval.
- The polygon, filled polygon.
- The string.
- The image of JPG, BMP.

13.2 How to draw images(JPG,BMP) on the drawing area

It is possible to draw images of JPG,BMP by the method: drawImage(), drawStretchedImage().

The method: drawImage() draws the image as is, and the method: drawStretchedImage() draws the image as specified size.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdrawingArea.h>
#include <WSCvslider.h>

void drawep(WSCbase* object){
    //drawing_a is same as newvdra_000...
    //You can get it extern WSCvdrawingArea* newvdra000; also.
    WSCvdrawingArea* drawing_a =
        (WSCvdrawingArea*)object->cast("WSCvdrawingArea");
    if (drawing_a == NULL){
        return;
    }
    WSCushort w = drawing_a->getProperty(WSNwidth);
    WSCushort h = drawing_a->getProperty(WSNheight);
    drawing_a->drawStretchedImage(0,0,w,h,"001.jpg"); // (A)
}
static WSCfunctionRegister op("drawep", (void*)drawep);
```

At (A), draws the image by size as same as the drawing area.

14 The indexed form

The indexed form switches the own area by its index tabs. At first, it requires to set the property: WSNmenuItems in order to define the index tabs. For example,if there are [tab1],[tab2],[tab3], set value as follows.

```
the property: WSNmenuItems
  tab1,tab2,tab3
```

Next,select one of the tabs and put instances on the selected area. It is possible to know which tab selected to see the property:WSNvalue. The above-mentiond example, it becomes the tab1:0, tab2:1, tab3:2.

15 The balloon help

15.1 How to indicate the balloon help

Drop an instance of the `WScvballoonHelp` class on the section `[NonGUI]` of the object box dialog on some window or some form, and specify the instance name to the property `WSNclient`, which instance indicates a balloon help. Specify the string to indicate to the property `WSNlabelString` of the balloon help instance.

16 The timer

16.1 How to use the timer

Drop an instance of the WSCvtimer class on the section [NonGUI] of the object box dialog on some window or some form, and set the interval of firing by milli second. It will fires the event WSEV_ACTIVATE after the interval. So set the event procedure with WSEV_ACTIVATE to the timer.

There are two kind of the timers. One fires only once, another continuously fires every interval. If the property:WSNcont is True, it will continuously fires. It is possible to start or stop the timer by the property: WSNrunning. If the property: WSNcont is False,after firing,the timer will stop, and the property: WSNrunning will become False.

17 The wizard dialog

The wizard dialog is used for making the interactive dialog. It has an indexed form internal which tabs are disavailable, and switch the scene by pushing the buttons [iBack] or [Nexti] in order.

At first specify the number of the scene to the property: WSNmenuItems.

The property: WSNmenuItems
the number of the scene

For example, set 5 to it,if the dialog has 5 scenes. Next about settings of the buttons, if you want the title of them are [iBack],[Nexti] on usual scene and are [iBack],[Finish] on last scene, set the following value to the proeprty: WSNlabelString.

The property: WSNlabelString
<Back,Next>,Finish

It is possible to see the property: WSNvalue which scene is indicated now. The first scene is 0,the next one is 1,2... And it is possible to specify the number to indicate a scene which you want to indicate. On editing the dialog, in order to indicate the scene on which you put instances, you must set the number of it to this property:WSNvalue.

The property: WSNvalue
The number of the scene
(Notice) This property must be specified the number of the scene
when you want to edit the scene on the application builder.

18 The controlling of the position of the instances by the offset

18.1 The offset of X,Y coordinate

It is possible to control the position of the instances of WSCvXXX class which has no window resource by the method: setXOffsetPtr and setYOffsetPtr.

For example, the following event procedure with initializing trigger shows setting the offset.

```
extern short xoffset; //A
extern short yoffset; //A
void initep(WSCbase* object){
    object->setXOffsetPt(&xoffset); //B
    object->setYOffsetPt(&yoffset); //B
}
```

At first,define a global short variable somewhere and do extern to access it at (A). set it as offset to the instance at B. Then the instance will be indicated at the position of the coordinate which is added the offset variable.

It is possible to set same variable to many instances and to control their position by it at once.

18.2 The scaling offset of the size of the intance

It is possible to control the size of the instances of WSCvXXX class by the method: setScaleOffsetPtr.

For example, the following event procedure with initializing trigger shows setting the scaling offset.

```
extern double scaleoffset; //A
void initep(WSCbase* object){
    object->setScaleOffsetPt(&scaleoffset); //B
}
```

At first,define a global double variable somewhere and do extern to access it at (A). set it as scaling offset to the instance at B. Then the instance will be indicated at the size which is multiplied by the scaling offset variable.

It is possible to set same variable to many instances and to control their size by it at once.

19 The memory device class

19.1 How to create the memory device class

It is possible to operate the data of the image directly by the memory device class. The memory device has the following facilities.

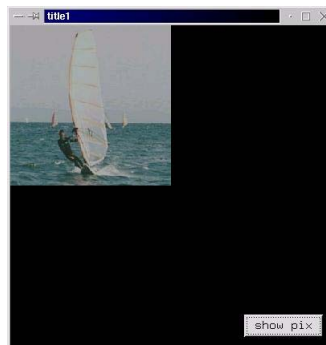
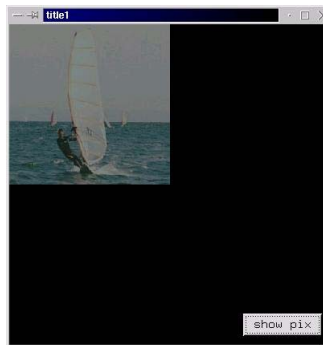
Drawing some pictures on the device.

Drawing some images(jpg,bmp) on the device.

Direct referencing and direct manipulation of the image data.

Transferring of the image data to the window.

By the sample: ws/sampes/share/memdev/newproject.prj, See the usage of the memory device class. This sample read the image "001.jpg" and indicates it gradually.





[The sample which indicates the image gradually.]

This sample does the following.

Creates two the memory devices(mdev,mdev2).

Draws the image "001.jpg" on mdev.

(1)Transfers the data changed brightness up from mdev from mdev2

(2)Transfers the data from mdev2 to the window.

Loops (1),(2).

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSDappDev.h>
#include <WSCcolorSet.h>
#include <WSCimageSet.h>
#include <WSCmainWindow.h>
extern WSCmainWindow* newwin000;

#include <WSDmwindowDev.h>
WSDmwindowDev* mdev = NULL;
WSDmwindowDev* mdev2 = NULL;

void btnep(WSCbase* object){
    WSDdev* dev = newwin000->getdev(); //A

    if (mdev == NULL){ //B
        mdev = WSDmwindowDev::getNewInstance();
        mdev2 = WSDmwindowDev::getNewInstance();
    }
}
```

```

mdev->createPixmap(200,200); //C
mdev->beginDraw(0,0,200,200); //D
WSDImage* image = WSGIappImageSet()->getImage("001.jpg"); //E
mdev->drawStretchedImage(0,0,200,200,image); //F
mdev->endDraw(); //G

mdev2->createPixmap(200,200); //H

mdev->initBuffer(); //I
mdev2->initBuffer(); //J

long i,x,y;
for(i=0;i<100; i++){
    for(x=0; x<200; x++){
        for(y=0; y<200; y++){
            WSCuchar r,g,b;
            mdev->getBufferRGB(x,y,&r,&g,&b); //K
            r = (WSCushort)((double)(r*i)/100); //L
            g = (WSCushort)((double)(g*i)/100); //L
            b = (WSCushort)((double)(b*i)/100); //L
            mdev2->setBufferRGB(x,y,r,g,b); //M
        }
    }
    mdev2->putBufferToPixmap(); //N
    mdev2->copyToWindow(dev,0,0,200,200,0,0); //P
}
}
static WSCfunctionRegister op("btnep",(void*)btnep);

```

It acquires the window device from the instance at (A) and creates the memory devices at the first click by the method: `getNewInstance` which creates an appropriate instance(B). It is impossible with new operator to create an instance of the memory device class, because this class depends the window system.

About indicating the image 001.jpg to mdev1, at first, initialize mdev1 by the method: `createPixmap` with the geometry (C). To begin draw pictures, call the method: `beginDraw()` of the device class(D). Acquire the image instance from the global image management class(E) and put the image instance to the memory device(F). If drawing pictures is over, call the method: `endDraw()`.

Initialize mdev2 too at (H). and initialize the memory buffer for direct operation (I)(J), this time it transfers the internal image data on the frame buffer to the memory buffer. At(K), get the RGB value from mdev1. At(L), increase the brightness of the RGB value and set it to mdev2 (M). It transfers the memory data to the frame buffer(N). and indicates it by transferring to window(P).

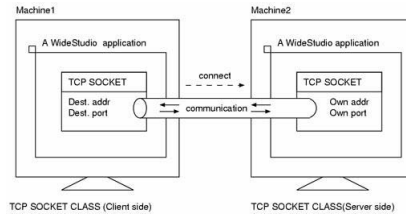
20 Network communication using TCP/IP

20.1 How to use network communication using TCP sockets

TCP network communication is used in client-server oriented communications. Server side socket, WSCvssocket class can accept from client side socket, WSCvsocket.

Usually, in handling TCP sockets by C/C++ language, connection is established after accept, listen or connect procedures but in WideStudio, the processes related to establishing TCP socket connection are automatically done in an object library to conceal these processes and users do not need to write these processes.

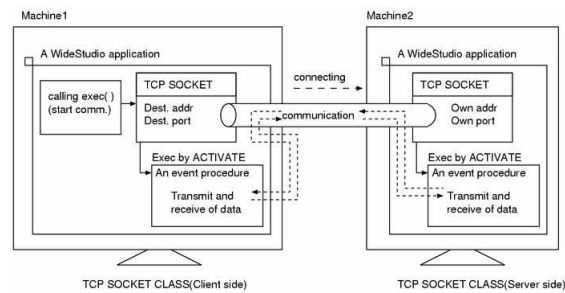
The TCP socket libraries have a property regarding IP address or PORT number, in which you can use just setting these values to transmit data using TCP socket libraries. TCP socket libraries contains connecting side(client) and connected side(server)



Client side and server side is different on operation mainly in connecting. Comparing a client side connecting at a specific IP address and port number existing over the network, a server side wait for being connected. Thus, in using TCP connection, WSCvsocket (Client side) should always connect to WSCvssocket (Server side) which is waiting for connection.

For the client side property settings, the referral server TCP/IP address is set in WSNip and the port in WSNport. For the server side property settings, the awaiting socket port is set in WSNport and turn "ON" WSNrunning. Though usually WSNip is not set, it should be set when the awaiting IP address have to be specified.

By invoking WSCvsocket::exec method in the client side, connection is established to the server side WSCvssocket. Once the connection is established, an ACTIVATE event arises on both client and server side to communicate in the event procedure.



Here is a sample of the event procedure starting in the ACTIVATE followed by success of connection which sends/receives client-side data.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvcsocket.h>

void com_ep(WSCbase* object){
    //do something...
    WSCvcsocket* sock = (WSCvcsocket*)object->cast("WSCvcsocket");
    char buffer[128];
    sprintf(buffer,"test!!! %d",cnt);
    cnt++;

    //send data;
    long send_len = sock->write((WSCuchar*)buffer,128);
    if (send_len == 128){
        //success! do something..
    }else{
        //error!
        return;
    }

    //receive data;
    buffer[0] = 0;
    long recv_len = sock->read((WSCuchar*)buffer,128);
    if (recv_len == 128){
        //success! do something..
    }else{
        //error!
        return;
    }
}
}
```



```
static WSCfunctionRegister op("com_ep", (void*)com_ep);
```

Next sample is the procedure for transmitting server-side data starting by ACTIVATE in establishing a connection.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvssocket.h>

void com_ep(WSCbase* object){
    //do something...
    WSCvssocket* obj = (WSCvssocket*)object->cast("WSCvssocket");
    char buffer[128];

    //receive data:
    //Store data from a client into buffer
    obj->read((WSCuchar*)buffer,128);

    //send data:
    //Store data for a client into buffer to send it
    strcpy(buffer,"send data...");
    obj->write((WSCuchar*)buffer,128);
}
static WSCfunctionRegister op("com_ep", (void*)com_ep);
```

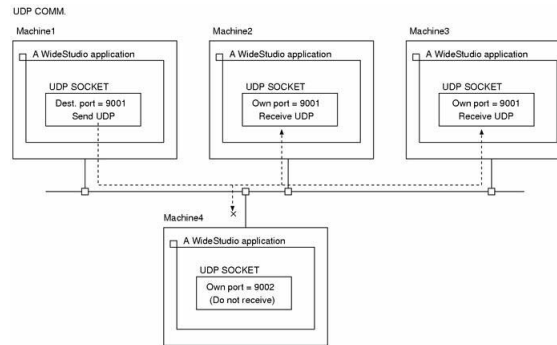
Next is an client-side event procedure which trasmits data after establishing connection.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvcsocket.h>
extern WSCvcsocket* newvcso_000;

void btnop(WSCbase* object){
    //Initiate connection and exchange data after connection
    long ret = newvcso_000->exec();
    if (ret != WS_NO_ERR){ //connection failed
        return;
    }
}
static WSCfunctionRegister op("btnop", (void*)btnop);
```

20.2 How to use a broadcast network communication using UDP socket

By using UDP, data can be sent to unspecified number of recipients (broadcast address: usu., xxx.xxx.xxx.255)
The data can be received by processes who wait in the port specified in sending.



For the setting in sending, referral socket port should be specified in WSNport. Also broadcast address, usually this is 255.255.255.255 is set in WSNip. In the system which is not allowed 255.255.255.255, the referral network address but with 255 in its last digit can be specified. For example, when you want to send to unspecified number of machines on the network:10.20.30.0, 10.20.30.255 can be specified.

For the recipient's property settings, the awaiting socket port is set in WSNport and turn WSNrunning "ON". As for the sender side, since connection is not being established, it becomes simple comparing TCP. It is as simple as sending data calling WSCvudpsocket::write.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvudpsocket.h>
extern WSCvudpsocket* newvudp_000;

void btnop(WSCbase* object){
    static long cnt = 0;
    WSCuchar buffer[64];
    //send data in buffer
    strcpy(buffer,"send data..");
    long ret = newvudp_000->write(buffer,64);
    if (ret < 64){
        //failed.
    }else{
        //success
    }
}
static WSCfunctionRegister op("btnop",(void*)btnop);
```

As for the recipient side, it receives in the event procedure which starts by ACTIVATE as the same as TCP's server side.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvudpsocket.h>
extern WSCvudpsocket* newvudp_000;

void recvop(WSCbase* object){
    WSCuchar buffer[64];
    //Receiving data
    newvudp_000->read(buffer,64);
}
static WSCfunctionRegister op("recvop",(void*)recvop);
```

21 Database access using the database class

21.1 Database access through ODBC

Using WSCvdb class enables to access a database through ODBC. In order to access ODBC, set WS_DB_ODBC in WSNTtype property and specify the DSN, the user name and the password.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    long ret = newvdb__000->open("dn","user","passwd");
    if (ret == WS_NO_ERR){
        //Connecting
    }else{
        //Connection failed with getting an error message.
        char buffer[1024];
        newvdb__000->getErrorMsg(buffer,1024);
    }
}
```

In order to access ODBC, DSN should set into WSNhostname, username in WSNusername, password in WSNpassword and call WSCvdb::open without arguments.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    long ret = newvdb__000->open();
    if (ret == WS_NO_ERR){
        //Connecting.
    }else{
        //Connection failure with getting an error message.
        char buffer[1024];
        newvdb__000->getErrorMsg(buffer,1024);
    }
}
```

21.2 Database access through PostgreSQL interface

By using WSCvdb class, you can access a PostgreSQL database directly through the PostgreSQL interface.

When accessing through the PostgreSQL interface, WS_DB_POSTGRES should be set in WSNtype and specify hostname, user name, password, database name, and port number into WSCvdb::open to call.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    long ret = newvdb__000->open("dn","user","passwd","dbname","5432");
    if (ret == WS_NO_ERR){
        //Connecting.
    }else{
        //Connection failure with getting an error message.
        char buffer[1024];
        newvdb__000->getErrorMsg(buffer,1024);
    }
}
```

In order to access PostgreSQL, the hostname that have the database should be set into WSNhostname, and user name in WSNusername, password in WSNpassword, database name in WSNdbname and port number in WSNport to execute WSCvdb::open without arguments.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    long ret = newvdb__000->open();
    if (ret == WS_NO_ERR){
        //Connecting.
    }else{
        //Connection failure with getting an error message.
        char buffer[1024];
        newvdb__000->getErrorMsg(buffer,1024);
    }
}
```

21.3 Creating the table

When the access to the database by WSCvdb::open is succeeded, you can issue SQL syntax to operate the database.

Next example shows how to create a table(shinamono) on the database

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    char buf1[512];
    strcpy(buf1, "create table shinamono(code int, hinmei char(20), nedan float)");
    newvdb__000->sqlExecute(buf1);

    if (ret == WS_NO_ERR){
        //Success
    }else{
        //Connection failure with getting an error message
        char buffer[1024];
        newvdb__000->getErrorMsg(buffer,1024);
    }
}
```

21.4 Store data in the table

You can store data into the table by issuing SQL syntax when connection to the database is succeeded by WSCvdb::open and there exists an operatable table.

Nex example shows how to store data in the table named shinamono on the database.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    newvdb__000->beginTran();
    strcpy(buf1, "insert into shinamono values(1, 'Orange', 100)");
    newvdb__000->sqlExecute(buf1);
}
```

```

strcpy(buf1, "insert into shinamono values(2, 'Apple', 200)");
newvdb__000->sqlExecute(buf1);
strcpy(buf1, "insert into shinamono values(3, 'Banana', 300)");
newvdb__000->sqlExecute(buf1);
strcpy(buf1, "insert into shinamono values(4, 'Melon', 0)");
newvdb__000->sqlExecute(buf1);

newvdb__000->commitTran();

}

```

21.5 Referring data on the table

You can store data into the table by issuing SQL syntax when connection to the database is succeeded by WSCvdb::open and there exists an operatable table.

Next example shows how to refer to/update the data on the table named shinamono on the database.

```

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
//Function for the event procedure
//-----
#include <WSCvdb.h>
extern WSCvdb* newvdb__000;

void db_ep(WSCbase* object){
    newvdb__000->beginTran();
    WSCdbRecord rs(newvdb__000);
    if(rs.open("select * from shinamono order by code") == WS_NO_ERR) {
        while (!rs.isEOF()) {
            rs.getColValue("code", &var);
            int code = (int)var;
            cout << "code:" << (int)var << " ";
            rs.getColValue("hinmei", &var);
            cout << "hinmei:" << (char*)var << " ";
            rs.getColValue("nedan", &var);
            char buf[80];
            double nedan = (float)var + 10;

            sprintf(buf, "%f", (float)var);
            cout << "nedan:" << buf << "\n";

            if(nedan != 0) {
                sprintf(buf1, "update shinamono set nedan = %f where code = %d", nedan, code);
            } else {
                sprintf(buf1, "delete from shinamono where code = %d", code);
            }
        }
    }
}

```

```
        newvdb__000->sqlExecute(buf1);  
        rs.moveNext();  
    }  
}  
rs.close();  
newvdb__000->commitTran();  
}
```


C The user defined classes

1 How to access to the member instances

1.1 How to access to the member instances in the class event procedures

To access to the member instance, it needs that the instance is defined as "member" of the user defined class. See [Class application window: capter[How to make a instance be member]]

The following shows a sample of a user defined class: sample.

The variable: `sample* base` is the pointer of the sample class. Then you can access to the member instance: `base->newlab000` which name is `newlab000` for example.

```
#include <sample.h>
void sample::event_procedure(WSCbase* object){
    sample* base = (sample*)object->getUserData(WC_BASE_CLASS);
    ...

    //Access the member instance...
    WSCvariant val = base->newlab000->getProperty(WCUserValue);
    ...
}
```

1.2 How to access to the member instances in the method

To access to the member instance in the class event procedure, it needs "base->", but in the method, it does no need "base->".

The following is the sample of a user defined class: sample. Then you can access to the member instance directly: `newlab000` which name is `newlab000` for example.

```
//a sample of the method.
void sample::method1(long data){
    newlab000->setProperty(WCUserValue,data);
    ...
}
```

D The store function

1 How to load the stored application window directly from the program

1.1 How to load the stored application window directly from the program

You can load the stored application window with following funtions.

The function for loading	Description
WSGFloadWindow(p1,p2,p3,p4)	Loading of the stored application window
char* p1	Specify the store Attributes: "FILE" = from the file.
char* p2	Specify the store data name: if p1 = "FILE",specify the file name
WSCbase** p3	Specify the pointer for returned value: the loaded a window
WSCbase* p4	Specify the parent instance for the partial application window
long (function returns)	WS_NO_ERR = success / the other = fail

The following shows to load the stored application window directly from the program.

```
#include "WSCconductor.h"
...

WSCbase* window = NULL;           //for returned value:
char* field = "FILE";             //from FILE.
char* fname = "newpic001.oof";    //FILE name.
char* path = "/usr1/win/data";    //specify the directory.

//Specify the directory.
WSGIconductor()->setSerializePath(path);

//Loading of the stored application window.
long ret = WSGFloadWindow(field,fname,&window,NULL);
if (ret == WS_NO_ERR){
    //success
    window->setVisible(True); //display the window.
}
```

Specify the storing attributes for the "field". The store function supports "FILE" now. In the future,it will support a RDB, an interprocess communication,a networking field, and so on...

Specify the directory for the "path" to the conductor which controls the storing function. The default value is the current directory. Specify the file name for the "fname".and you can load the stored application window with WSGFloadWindow(). WSGFloadWindow() creates the instances when if it succeeds, so if you call it several times,it creates several instances.

Please check the directory,the file name,the permission of the specified file,if it fails.

1.2 How to load the stored partial application window directly from the program

The following shows to load the stored partial application window.

```
#include "WSCconductor.h"
...

WSCbase* window = NULL; //for returned value:
WSCbase* parent = newwin000; //the parent instance for
//the partial application window.
char* field = "FILE"; //from FILE.
char* fname = "newpic001.oof"; //FILE name.
char* path = "/usr1/win/data"; //specify the directory.

//Specify the directory.
WSGIconductor()->setSerializePath(path);

//Loading of the stored partial application window.
long ret = WSGFloadWindow(field,fname,&window,parent);
if (ret == WS_NO_ERR){
    //success
    window->setVisible(True); //display the window.
}
```

It is same as the loading of the stored application window fundamentally, but it requires the parent instance in this case, for the partial application window, because it is not a window and needs a parent window to display. You can specify the parent window like, WSCwindow, WSCform, WSCscrForm which has the management function of child instances.

2 How to delete the loaded application window

2.1 How to delete the loaded application window

The following shows to delete the loaded application window directly from the program.

```
//delete the application window
WSGFdestroyWindow(window);
```

Specify the application window to delete. You can delete the partial application window or the normal instance also, but you can not delete the same instance twice because it causes memory fault. After deleting, you can not access the deleted instance.

2.2 How to delete the loaded partial application window

The following shows to delete the loaded partial application window directly from the program.

```
//delete the partial window
WSGFdestroyWindow(object);
```

It is same as the loading of the stored application window fundamentally, but you specify the partial application window in this case,

E The remote instance function

1 Accessing a remote instance

1.1 Accessing a remote instance

You can get a remote instance by requesting to the object management instance which exists one per a load module.

Object management class	Instance retrieving class
WSCbaseList	WSCbaseList* WSGIappObjectList()

The way to retrieve the object to access is as follows:

```
#include "WSCbaseList.h" //Access WSGIappObjectList()
#include "WSCRbase.h"    //Use the virtual remote instance class
...
void event_procedure(WSCbase* object){

    //Get WSCRbase point by the object management
    char* obj_name = "newvlab_001";    //Remote instance named newvlab_001
    WSCRbase* rinstance = WSGIappObjectList()->getRemoteInstance(obj_name);

    //Access to the remote instance by the virtual remote instance
    rinstance->setProperty(WSNlabelString,"HELLO WORLD");
```

rinstance is the instance to access the remote instance. It requires the object name as its argument. You can access remote instances through the virtual remote instance as well as you can access usual instances(objects)

1.2 Casting a remote instance

You can use the virtual remote instance acquired through the object management by casting to original class as well as usual objects. Casting is required when you need to call methods existing in the original class. Next example is to cast WSCRlist type virtual remote instance in order to call WSCRlist::addItem() from WSCRbase type virtual remote instance.

```
#include "WSCbaseList.h" //Access to WSGIappObjectList()
#include "WSCRlist.h"    //Use the virtual remote instance class
...
void event_procedure(WSCbase* object){

    //Getting a WSCRbase pointer by the object management
    char* obj_name = "newlist_001";    //Remote instance named newlist_001
    WSCRbase* rinstance = WSGIappObjectList()->getRemoteInstance(obj_name);

    //Cast WSCRlist class remote instance to the virtual remote calss WSCRlist that corresponds
```

```
//to the original WSClist class
WSCRlist* rlist = (WSCRlist*)rinstance->cast("WSCRlist");
if (rlist == NULL){
    //Not WSCRlist class
    return;
}

//Call a WSClist class method through WSCRlist virtual remote intance class

rlist->addItem("item..");
```

F Samples and demonstrations

1 Sample:1 (Hello World)

Here, you create a project, an application window and event procedures. The following is a rudimentary sample which displays "HelloWorld" by pushing the button of the window,

The source code is ws/samples/EE/hello/hello.prj. You can load the project and build by the application builder.

- Creating a project
Select the [New project] of the [Project] menu, and input "hello" for the project name, and check the [Normal application].
- Creating a application window
Select the [New window] of the [File] menu, and Check the [Normal window], and check [Add to project], and input the window name "newwin000".
- Placing the instances
Place the WSCvbtn instance on the created application window. Select the [ObjectBox] of the [View] menu, to display the object box dialog.
The next, clicking the [Commands] tab on the object box dialog, and dragging and dropping the WSCvbtn class shown as BTN from the dialog to the application window, it creates the new instance of WSCvbtn.
you can adjust the properties of the instance by the inspector.



[A view of the application window]

- Creating the event procedure
At first, select the button instance by the inspector, and select [New procedures] of sub-menu: [Procedures] of [Edit] menu.
Here, set "btn_proc" for the function name and WSEV_ACTIVATE trigger.
To create a template file of the event procedure, push the button [Template].
- Editing the event procedure
The next, you edit the template event procedure. you can execute the source code editor by selecting [Edit] of sub-menu: [Procedures] of [Edit] menu to edit the event procedure which is focused on the inspector. Here, the following procedure shows that it shows "HelloWorld" at the first clicking, it exits the application at the second clicking.

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----
```

```
//Function for the event procedure
//-----
void btn_proc(WSCbase* object){
    //do something...
    static long cnt = 0;
    if (cnt == 0){
        object->setProperty(WSNlabelString,"Hello World.");
        cnt++;
    }else{
        exit(0);
    }
}
static WSCfunctionRegister op("btn_proc", (void*)btn_proc);
```

- Saving the project
You can save the project by [Save project] of [Project] menu.
- Building the project
You can build the project by [Build all] of [Build] menu. After building, execute the application!



[Executing the application]

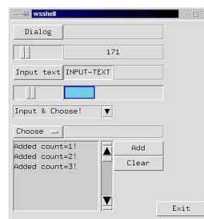
2 Sample:2(Various kinds of classes)

Here,the following show the instances of the various kinds of WideStudio classes.

The source code is ws/samples/EE/sample/sample.prj. Please load it and build it by the application builder.

this sample shows the following things.

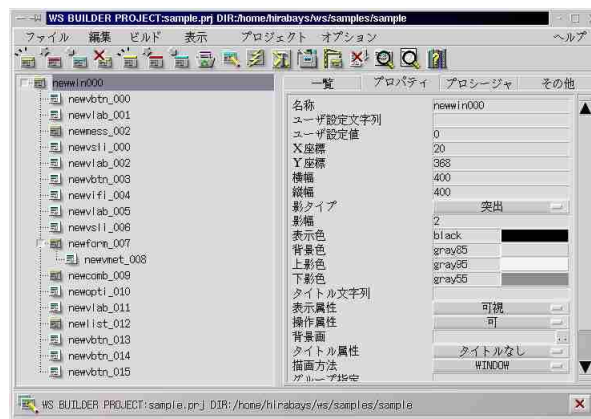
- To display sample dialogs.
- To display value by the slider.
- To display the text input demonstrations.
- To display the combobox demonstrations
- To select a value by option menu.
- To display a list of texts.



[A view of the application window]

- To display the dialogs
The button which displays "Dialog" has a event procedure which trigger is WSEV_ACTIVATE and which function is btn1_ep(). This procedure is executed by clicking of the button,and pops up the message dialog: [newmess_002] and displays the return value from it on the label:[newvlab_001].
- To display the value of the slider
The slider: [newvsl_000] has a event procedure which trigger is WSEV_VALUE_CH and which function is slider1_ep(). This procedure is executed by sliding of the slider,and displays the slider value on the label:[newvlab_002].
- To input strings
The button which displays "Input text" has a event procedure which trigger is WSEV_ACTIVATE and which function is btn2_ep(). This procedure is executed by clicking of the button,and gets the string from the input field:[newvfi_004] and display it on the label:[newvlab_005].

- To display the value of the slider (2)
The slider2: [newvsl_i006] has a event procedure which trigger is WSEV_VALUE_CH and which function is slider2_ep(). This procedure is executed by sliding of the slider, and displays the slider value on the meter: [newvmet_008].
- A demonstration of the combo box
It shows the demonstration of the combo box.
- A demonstration of the option menu
The option menu which displays "Choose" has a event procedure which trigger is WSEV_VALUE_CH and which function is opt1_ep(). This procedure is executed by choosing of the menu, and displays the choosed value on the label: [newvlab_011].
- A demonstration of the list
The button which displays "Add" has a event procedure which trigger is WSEV_ACTIVATE and which function is btn3_ep(). This procedure is executed by clicking of the button, and adds a string to the list: [newlist_012].
The button which displays "Clear" has a event procedure which trigger is WSEV_ACTIVATE and which function is btn4_ep(). This procedure clears the list: [newlist_012].



[A view of the application window]

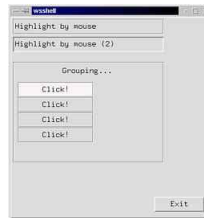
3 Sample:3 (label)

The following is a demonstration of the event procedure about labels.

The source code is ws/samples/EE/labelwork/labelwork.prj. Please load it and build it by the application builder.

this sample shows the following things.

- To highlight the label by mouse implemented by normal event procedures.
- To highlight the label by mouse(2) implemented by an initialize procedure and sub-procedures.
- To group the labels by an event procedure



[A view of the application window]

- To highlight the label
The button which displays "highlight by mouse" has two event procedures with WSEV_MOUSE_IN trigger and WSEV_MOUSE_OUT trigger.
The WSEV_MOUSE_IN procedure stores the original bgcolor to WSNuserString and stores the highlight color to WSNBackColor.
The WSEV_MOUSE_OUT procedure gets the original bgcolor from WSNusreString and stores it to WSNBackColor.
- To highlight the label (2)
The button which displays "highlight by mouse(2)" has an event procedures with WSEV_INITIALIZE trigger which setups sub-procedures which trigger is WSN_MOUSE_IN and WSN_MOUSE_OUT.
The initialize procedure like this sipmlifies because it not require that the instance has many pro-cedures for one-function.
- To group the labels by an event procedure
The button which displays "Click!" has a event procedures which trigger is WSEV_MOUSE_PRESS.
This procedure make a group of the labels which has it by storing selected label to the parent instance : form or window. Then the labels highlights cooperated.

4 Sample:4 (A calculator)

This is a demonstration of calculator using by the array of the buttons.

The source code is ws/samples/EE/labelwork/wscal.c.prj. Please load it and build it by the application builder.

this sample shows the following things.

- To demonstrate calculating with the operator: +, -, /, *
(Notice)The specification of the calculator is not strict.



[A view of the application window]

- The number button
Input the number to the indicator.
- The operator button
Stores the operator if the button is "+", "-", "/", "*", Calculates by the stored information if the button is "=",
- The other
The resize event procedure adjusts size/place of the buttons when the window is resized.