WideStudio Application Builder User's Guide

Contents

Chapter 1

Introduction

1.1 What is the WideStudio?

WideStudio is a GUI based application development environment which has been developed purely in Japan, on many platforms such as Windows, Linux, FreeBSD, and Solaris. This application is fully available and is "open source". This is one of its most outstanding features. Everyone can use this to develop open and multi-platform applications with ease.

WideStudio has the following characteristics:

- Open source License (X11/MIT).
- C++ based development.
- Possible to build platform-independent GUI applications.
- Uses fully original class libraries.
- Equipped with the application builder which can easily edit a GUI application.
- Needs only a minimum of coding, in an event-driven style.
- Ensures scalability by a function of building / importing additional class libraries.

Mature developers may have developed applications on multi-platforms such as Windows and X11.

I have also developed various applications by saving money, getting a palmtop computer, inserting the machine language on it by drawing dots to make a so called game, but that is an old story. Since then, I used MS-DOS followed by Windows 3.0, then 3.1, UNIX/X11 Window system, Xt/Motif, Windows95/NT(Win32), MFC/ActiveX.

At last, I don't want to bother anymore with incompatibility between platforms and have decided to make my applications platform-independent.

I thought it be great if developers could build applications by only using a mouse with the WideStudio IDE. The next figure describes the composition of WideStudio and its files.

[WideStudio Composition]

1.1.1 What is the C++ language?

WideStudio is written in C++. C++ is basically the C programming language, with enhanced support for object oriented programming.

C is the one of the most standard programming languages, and is used in many larger software products because of its hardware accessing ability. C++ inherits many features from the C language, and is equipped with functions to allow developers to build applications more efficiently.

While it copies C format, it soon becomes apparent that it is a world apart from C, and tends to get both novice and experienced programmers into difficulty. Using WideStudio, developers can write programs as they did in C, and they are still able to make good use of great C++ functionalities, and of course, programs using classes are also easy to make.

1.1.2 Comments on the development of the WideStudio

There are thousands of application builders like WideStudio, so why use WideStudio?

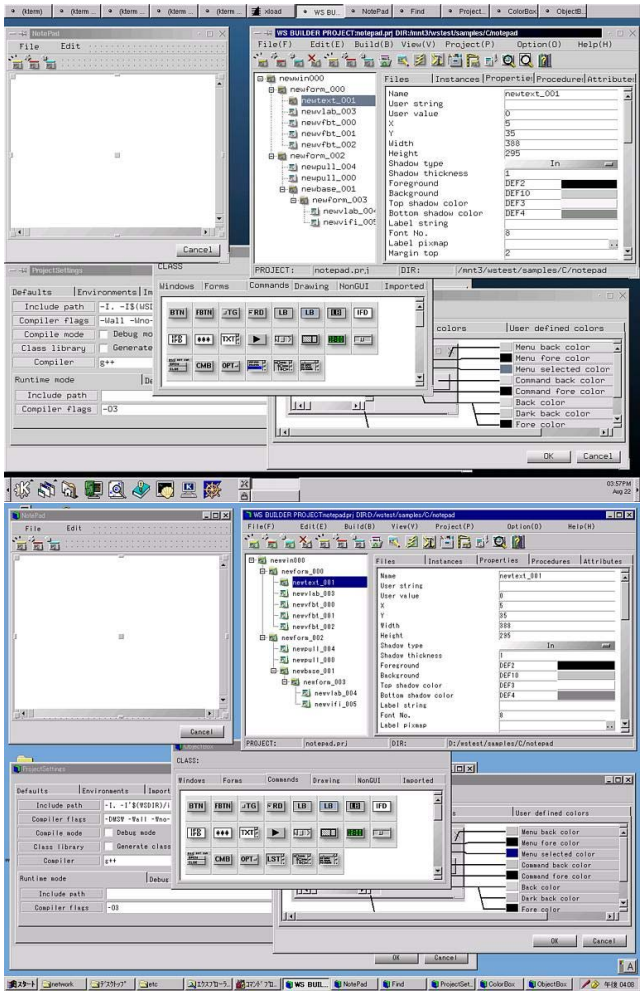
I have often heard "I want to make some applications, but it's hard...", or "I don't want to buy a shit application builder and pay money for it!" and they cannot take up programming. WideStudio solves these problems. It's free, and so easy to use!

I also often hear "What the hell is open source?" and "Why is the WideStudio free?" One of the objectives of developing WideStudio is to contribute it to the open source world. WideStudio was made in a restricted time frame to see how efficiently it could be built / designed. I would appreciate if any part of this source would be helpful for developers in developing or designing software from now.

From nuts and bolts to real applications, it will be a huge delight if everyone shares enthusiasm in developing through engineering.

1.2 Multi-Platform Environment

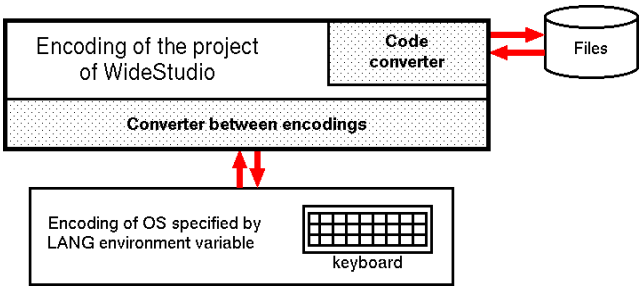
WideStudio enables you to easily develop multi platform applications without being aware of platforms themselves, in a highly consistent environment. This means that an application developed on Linux can be also run on Windows.



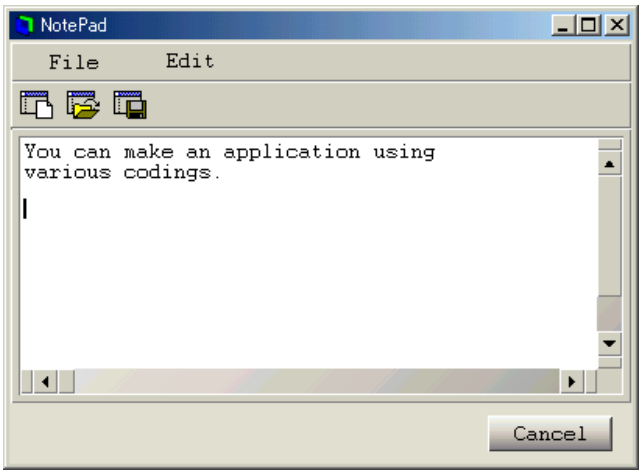
[WideStudio on Unix / Linux (top), WideStudio on Windows (bottom)]

1.3 Multi Encoding

WideStudio supports many coding systems such as Unicode(UTF8), EUCJP, and SJIS. WideStudio can deal with the encoding independently to the encoding on each platform so; for example, EUCJP applications built on Linux can also run on Windows with EUCJP codes without any modifications. By specifying the LANG environment variable, WideStudio understands the coding system that the system uses for the keyboard input, and converts automatically. It also enable us to transform coding systems for external data file I/O.

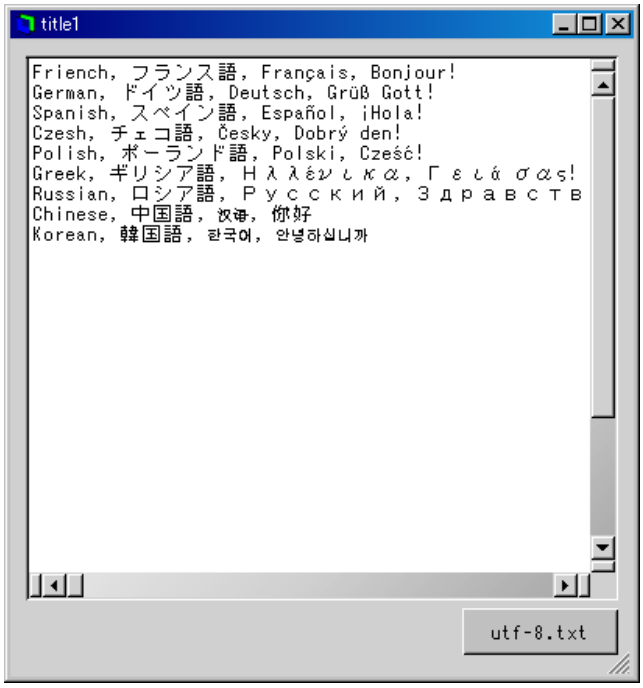


[WideStudio Encoding]



[Example of displaying EUCJP code on Windows]

Also, users can develop an internationalized application using Unicode.



[Displaying an Unicode text in some languages]

WideStudio supports the following encoding systems.

The encoding	
ISO8859_1	European
ISO8859_2	Eastern European
ISO8859_3	Turkish,Esperanto
ISO8859_4	Baltic
ISO8859_5	iso8859-5
ISO8859_6	Arabic
ISO8859_7	Greek
ISO8859_8	Hebrew
ISO8859_9	Turkish
ISO8859_10	Baltic2
ISO8859_13	ISO8859(13)
ISO8859_14	ISO8859(14)
ISO8859_15	European 2
UTF8	UNICODE UTF8
KOI8R	KOI8 Russian
EUCJP	EUC Japanese
SJIS	SJIS Japanese
EUCKR	EUC Korean
EUCCN	EUC Chinese
BIG5	BIG5 Chinese

1.4 Open Source License

WideStudio is an open source application. Everyone can access the source codes of WideStudio, and re-distribute it freely.

WideStudio can be used for business also with no charge because of the X11/MIT license.

- Free re-distribution of the source code
- X11/MIT License
- Free commercial use
- No guarantee

1.5 Downloading of WideStudio

WideStudio is available for download from the following locations:
http://www.widestudio.org/ (original site)
http://www.sourceforge.jp/projects/widestudio/
http://www.sourceforge.net/projects/widestudio/
http://www.vector.co.jp/

1.6 Building and installing WideStudio

1.6.1 Building and installing on a UNIX system

Start by downloading the source code and online manual from the WideStudio web site; unpack it as follows.

Notice: vX.X stands for the version of the WideStudio.

On Linux:

```
cd /tmp
tar -zsxvf ws-vX.X.tar.gz
cd /usr/local/ws
tar -zsxvf ws-vX.X-doc.tar.gz
```

On Solaris:

```
cd /tmp
gzip -cd ws-vX.X.tar.gz | tar -moxvf -
cd /opt/ws
gzip -cd ws-vX.X-doc.tar.gz | tar -moxvf -
```

There are several targets to build in WideStudio, such as the runtime libraries, the debugging libraries and so on. The runtime libraries are for normal use, and the debugging libraries are for debugging applications or developing the WideStudio.

Notice: "ws" in the examples below indicates the directory where the WideStudio has been installed.

Notice: Sometimes compile errors will occur due to differences in the machine environment. In such case, edit the file: ws/sys/config/mkfiles to configure the

compiler flags, the include path, the library path and so on, to avoid compile errors.

In the case of FreeBSD, use the command "gmake" instead of "make".

To build all:

```
cd ws/src
./configure
make
```

To build the runtime libraries and the application builder:

```
cd ws/src
./configure
make runtime
```

To build only the runtime libraries for debugging:

```
cd ws/src
./configure
make debug
```

After the building, make sure that the application builder is in ws/bin/. Then execute the following command as super user to install it.

```
cd ws/src
make install
```

1.6.2 Installing with the binary RPM package on the Linux operating system

Install with the binary rpm package is easy. It does not require building the source code. Execute the following command as super user to install it.

Notice: vX.X stands for the version of the WideStudio.

```
rpm -i ws-runtime-vXXXX.i386.rpm
```

WideStudio will be installed in the directory /usr/local/ws. The rpm package of the WideStudio requires some C++ runtime library rpm package. If you get some error like [libstdc++-libc6.1-1.so.2 not found], you have to install it too.

1.6.3 Installing on Windows operating system

Start by unpacking the downloaded zip file; you get a directory "wsinst". In order to install, execute "setup.exe" in that directory. The following wizard dialog will show up, then specify the install directory.

[The setup wizard dialog of the WideStudio]

Next, specify your favorite Web browser.

[Settings of the Web browser]

Next, specify your favorite source code editor.

[Settings of the source code editor]

Next, it indicates the license agreement of the WideStudio and GNU, and installing is done. Reboot your computer (this is necessary).

1.7 Settings After Installation

1.7.1 Settings on Windows

The installer of the WideStudio sets up environment variables; no user interaction is required, except for rebooting after installation of the WideStudio in order to refresh the environment.

1.7.2 Setting the environment variables on a UNIX system

The following environment variables have to be set up to get the WideStudio ready after installing it.

WSDIR	Installation directory of the WideStudio
PATH	Binary directory of the WideStudio
LD_LIBRARY_PATH	Shared library directory of the WideStudio

If the environment variable "WSDIR" is not set, the application builder treats it as having the following value.

OS	Default value
Windows	C:/Program Files/WideStudio/ws/
Linux	/usr/local/ws/
SunOS	/opt/ws/
other UNIXs	/usr/local/ws/

For example, in case the installation directory is "/export/home/ws", set the environment variables as follows.

Note: xxxxx in the list shows the value before modification.

in csh	setenv WSDIR /export/home/ws setenv PATH xxxxxx:/export/home/ws/bin setenv LD_LIBRARY_PATH xxxxxx:/export/home/ws/lib
in sh	WSDIR="/export/home/ws"; export WSDIR PATH="xxxxxx:/export/home/ws/bin"; export PATH LD_LIBRARY_PATH="xxxxxx:/export/home/ws/lib" export LD_LIBRARY_PATH

Add these settings above to the file ".cshrc" (for use with csh) or ".profile" (for use with sh), and reflect it by logging out or executing rehash command (in csh) to update the values. See the shell manuals for the details of setting of shell environment variables.

1.7.3 The other files on a UNIX system

The files which are used by the application builder are as follows. These files are automatically created by the application builder in the home directory.

File name	Description
.wsrc	Settings of Look and Feel, default colors.
.wsbuilderrc	Editor, browser
.wsbuilder_defaults	Information of the application builder

The "wsreset" command can be used to reset these files in order to initialize the settings when some files have wrong values.

1.8 Uninstalling WideStudio

1.8.1 Uninstalling on Windows

WideStudio can be uninstalled by selecting it in the Add / Remove Application dialog of the control panel. One can also uninstall it by executing "unsetup.exe".

```
ws/bin/unsetup.exe
```

1.8.2 Unintalling on a UNIX system

To uninstall the WideStudio, use "make uninstall" if the WideStudio was built by "make" and installed by "make install". Change directory to the building directory of the WideStudio, and execute "make uninstall" as super user.

```
make uninstall
```

1.8.3 Uninstalling the RPM package on a Linux system

If the WideStudio was installed from the binary rpm package, execute the following command as super user to uninstall the WideStudio.

```
rpm -e ws
```

Chapter 2

Getting started

2.1 What is the application builder?

The application builder is very strong and usable program, that draws out the maximum ability of WideStudio, which is an integrated development environment (IDE) for GUI applications. Application developers are released from complicated implementations of GUI applications by using the application builder of the WideStudio.

Specification of the application is as follows.

- Making and managing the application window

  Making the application window with interactive operations, and their management is done automatically.

- Implementation with the event procedure

  Minimum coding of applications by an event driven system. Developers prepare the event procedure and set them to the GUI objects.

- Properties of the objects

  Developers can modify the properties of the objects with interactive operations.

- Informations of the application

  Developers can refer the objects (the instances) and the event procedures, their properties, and other information by the application builder.

- Management as project

  Management of application window files and event procedure files, is part of the environment for development and building by project unit.

- Automatic building of the application

  It generates the makefile and c++ source codes of the application window, compiles and links them, and creates the executable.

- Executing and debugging of the application

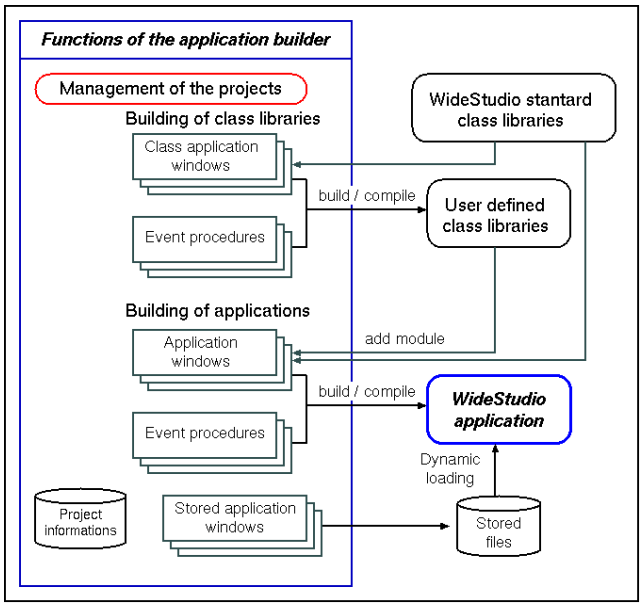
  It executes and debugs the built application.

- Supporting of the class application window The class application window is used to build the application window as a new c++ class of the WideStudio. The application builder generates the class library by integrating the classes of the WideStudio, and makes it possible to use it from other applications.

- Support of the stored application window

  The application builder is used to store whole or part of the application window to a file. It is also possible that applications can load it dynamically and share it while they are running. Treating objects as stored application windows is effective for applications which display some maps, drawings, and wiring diagrams.

The following shows the functions of application builder, and structure of the files.



[Outline of the application builder]

2.2 Development of an application on WideStudio

The method of development by WideStudio Application Builder is as follows.

- Creating a project

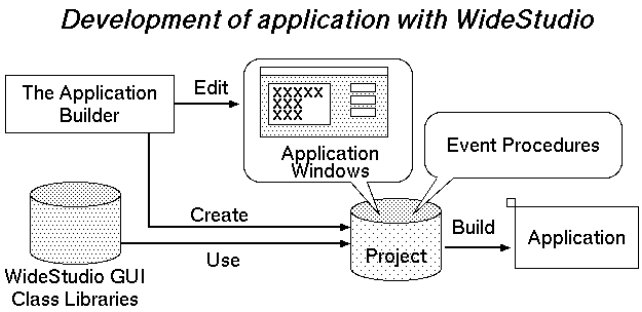
  First the developer creates a project for an application. A project is an unit of application development. For detail, see Chapter 6.

- Creating the application window

  The developer can develop application windows visually with the WideStudio Application Builder. For detail, see Chapter 3.

- Creating the event procedures

  The event procedure is executed when an event occurs. The developer implements easily the procedures which do some advanced processes, such as complicated painting and data processing, by C/C++ language, and can chose the event to execute it. For detail, see Chapter 5.

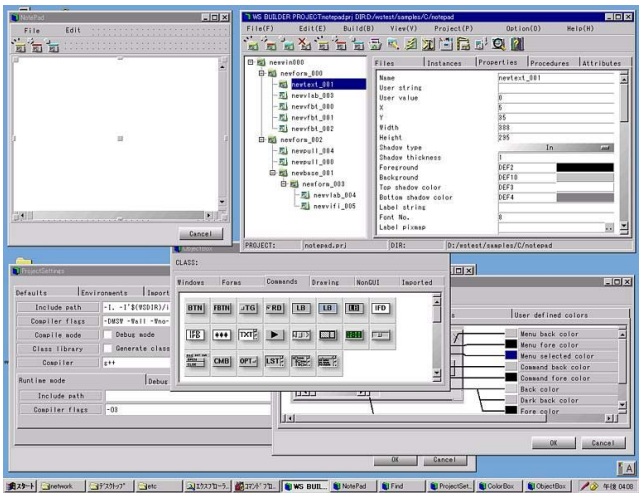


[The method of development of an application]

2.3 How to start the application builder

2.3.1 How to start on Windows

The user can start the application builder by clicking the wsbuilder icon, or selecting wsbuilder in the start menu.

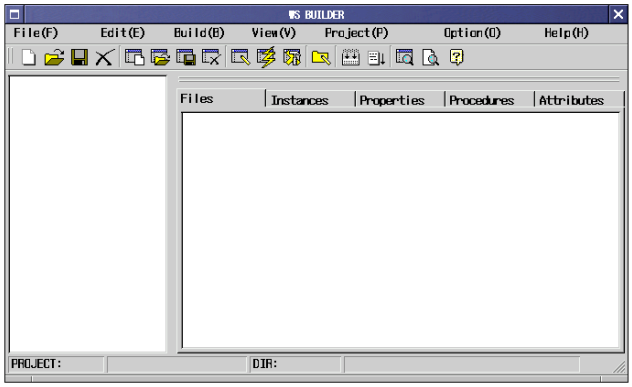


[The application builder on Windows]

2.3.2 How to start on the UNIX system

The user can start the application builder with wsbuilder command on the X-Window system.

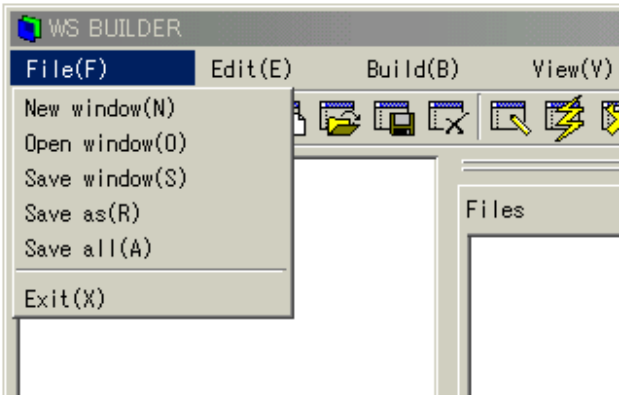
It requires the C++ compiler, debugger, and source code editor to build applications.



[The initial view of the application builder]

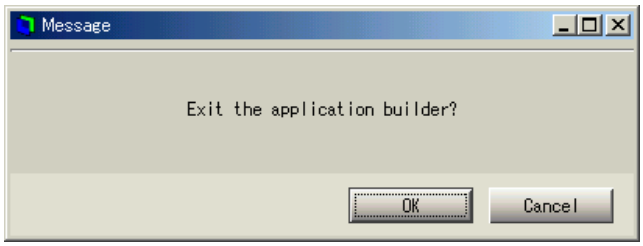
2.4 How to exit the application builder

Select ((menu:File → Exit)) to exit the application builder, or push the [X] button at the top right of the window.



[Exit the application builder]

The following dialog pops up, and then click its [OK] button. If there are unsaved application windows or projects, the question dialog pops up to confirm saving them.



[The exiting dialog]

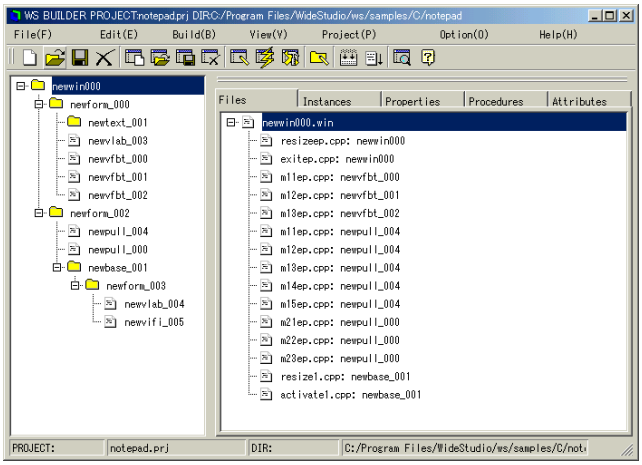
2.5 The name and function of the parts of the builder

With the application builder, the developer can configure the window of an application visually. The builder has the following panes, parts and windows.

- Inspector
- Source code viewer
- Property editor
- Event procedure editor
- Window attribute editor

2.5.1 Inspector, Source code viewer

The developer can confirm the structure of the instances on the application window in the Inspector. In addition by Source code viewer, the developer can see which instance has source codes for the event procedure.



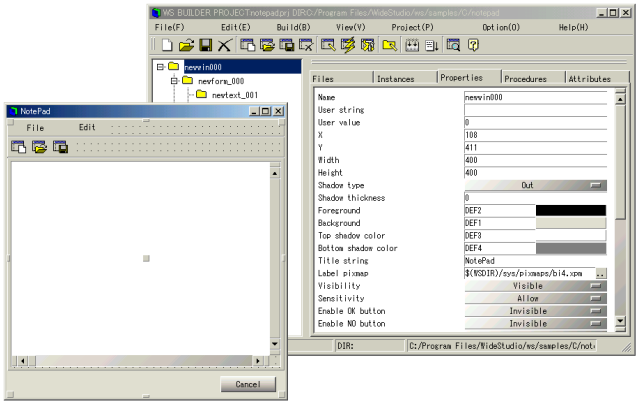
[The structure of the instances (left) and source codes (right)]

Clicking an instance in the Inspector, it is selected.

In the case [+]/[-] mark appears in the Inspector, the instance has child instances: by double clicking, it opens the instances and the children is shown on the instance tree.

2.5.2 Property editor

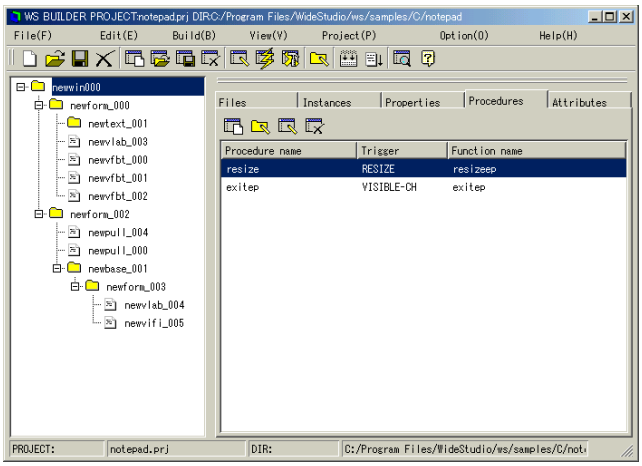
The instance of the WideStudio has internal variables as the properties which define the geometry, colors, shapes, types, and so on. Also these properties can be changed visually by Property editor of the application builder.



[Changing values of the property by Property editor]

2.5.3 Event procedure editor

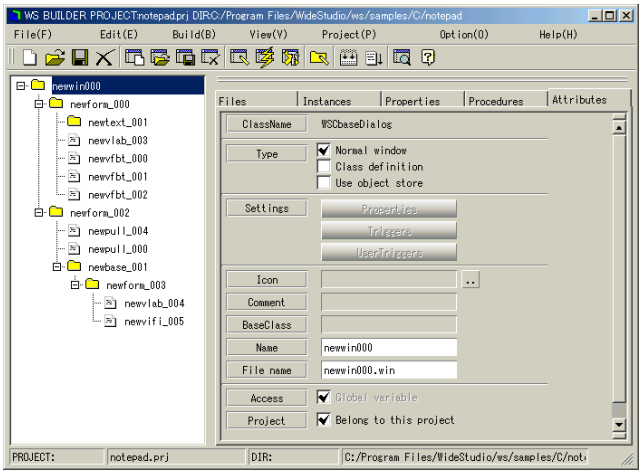
On the WideStudio, programming for event driven system is required. The developer puts some event procedures into instances with triggers. For example, an event procedure is associated with a button instance with ACTIVATE trigger. Then, clicking the instance activates the trigger is activated and the procedure is executed.



[Add/edit/delete the event procedures by Event procedure editor (right)]

2.5.4 Window attribute editor

An instance has some attributes like instance definition type, and so on. The developer can edit them by Window attribute editor.



[Editing the attributes by Window attribute editor (right)]

2.6 Development of a simple application 'Hello'

Let's develop a simple application "Hello" with the WideStudio Application Builder. The application has a push button, and pushing it displays "Hello!" in the button. The development of the simple application "Hello" supports to understand the development flow of WideStudio.

[A simple application "Hello"]

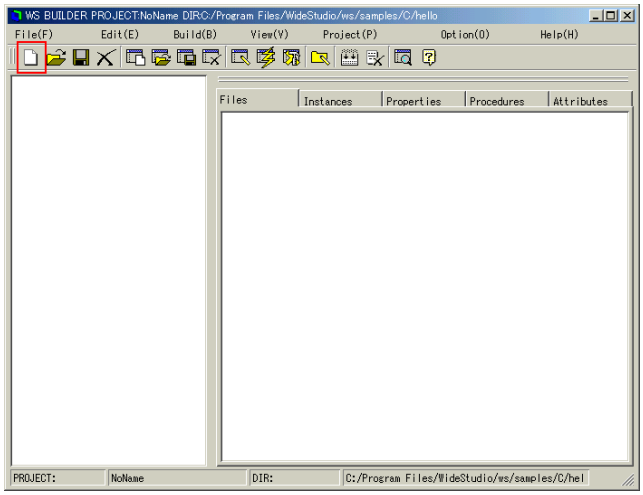
For the application "Hello", follow the flow as followings.

- Creating a project for "Hello".
- Creating an application window for "Hello".
- Creating an event procedure to indicate "Hello!".
- Building the application "Hello".

2.7 Creating a project 'Hello'

Now create the project for the application "Hello". To bring up the wizard dialog by ((menu:Project → New project)) or by clicking the icon shown in the next figure.

In the wizard dialog, input the project name (the application name) "hello", directory name where files are saved, and chose [Normal application] type.

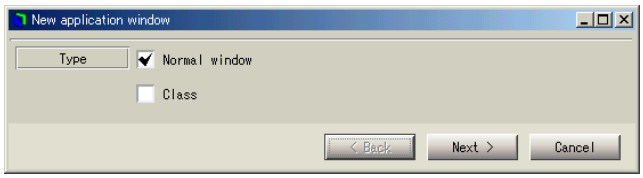


[Creating a project for the application "Hello"]

2.8 Creating an application window

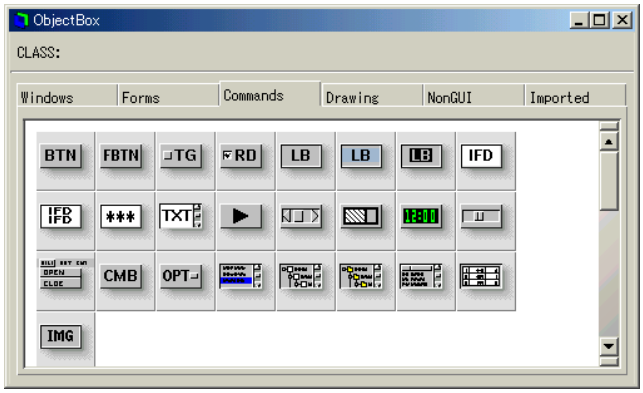
On WideStudio, the window developed by user is referred as the "application window".

Now create an application window for the application "Hello". Select ((menu:File → New window)) and chose [Normal window] for the window type in the wizard dialog. Then, input the application name (default is "newwin000") in it and create window with "None" template.



[The wizard dialog for creating application window]

The ObjectBox dialog is used to put GUI instances onto the application window. In order to indicate it, select ((menu:View → ObjectBox)), then a dialog will appear as in the following picture.



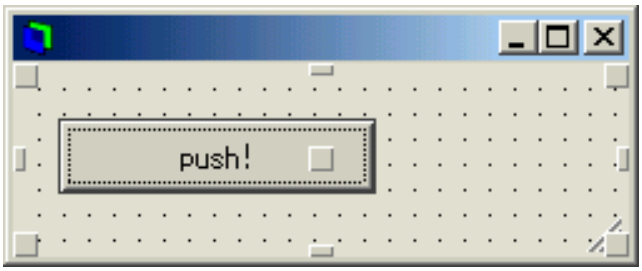
[The ObjectBox window]

In order to drag and drop a push button instance to the window, click the [Commands] index tab in the ObjectBox window, and drag the icon [BTN] (WSCvbtn/push button class) and drop it into the application window as in the following picture.

It is possible to edit the properties of the instance with the application builder. So let's change the size of the push button instance. Select the instance, click the [Properties] tab on the application builder, and set properties as follows.

- Name: newvbtn_000
- X: 10
- Y: 10
- Width: 200
- Height: 30
- Label string: push!

It becomes as in the following picture.



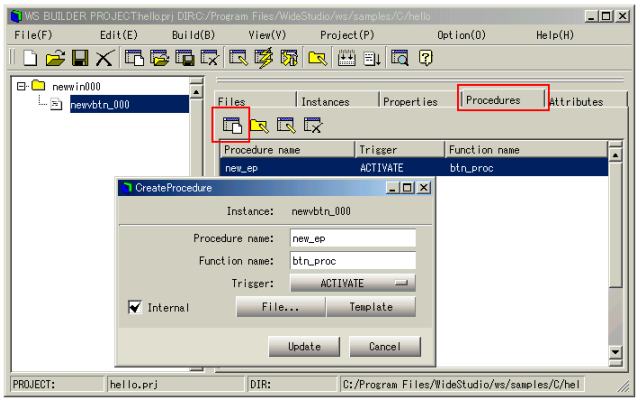
[The application window "Hello" under construction]

2.9 Creating an event procedure.

Instances triggers various kinds of events under execution. On WideStudio, it is possible to put some event procedures on the instance so that they will be executed with the event triggered. For example, let's create an event procedure which has the following function.

- Indicating "Hello!" on the button by clicking it on the window.

Set an event procedure to the instance so that it executes the procedure by clicking the instance: newvbtn_000. At first, select the instance: newvbtn_000 in the Inspector, then select the [Procedures] index tab of the application builder, and click the leftmost icon to create an event procedure.

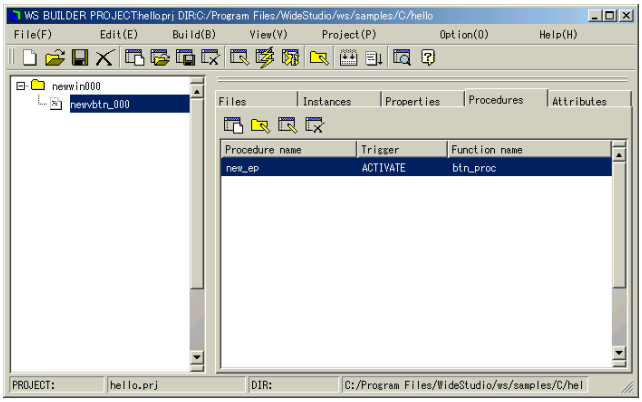


[Creating an event procedure]

Input the procedure name, the function name, and the others in the dialog as follows.

- Procedure name: new_ep
- Function name: btn_ep
- Trigger: ACTIVATE

The procedure name is used to identify the event procedure, therefore it can be any string including spaces. The function name is a name of C/C++ function for which the developer writes codes. The last one is the trigger (ACTIVATE will be executed normally by clicking the button). A template file of the source code for the event procedure is created.



[A event procedure]

By clicking the event procedure created as above, the source code editor appears with the following template of the code for the event procedure.

The developer can specify the default editor. Select ((menu:Project → Project settings)) and input the path of the editor in the "project settings". The default editor is vi on the UNIX system and notepad.exe on the Window system.

```
#include <WScom.h>
#include <WSCfunctionList.h>
```

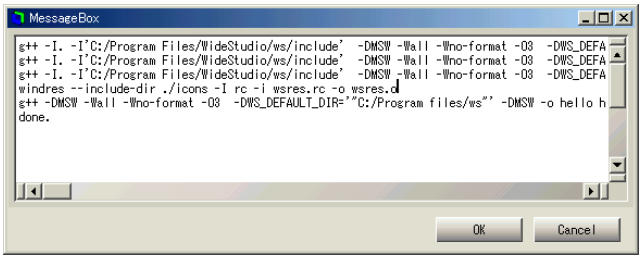
```
#include <WSCbase.h>
//-----------------------------------------------
//Function for the event procedure
//-----------------------------------------------
void btn_ep(WSCbase* object){
  object->setProperty(WSNlabelString, "Hello!"); //A
}
static WSCfunctionRegister op("btn_ep",(void*)btn_ep);
```

The WSCbase* pointer "object" denotes the instance in this code. At //A, the string "Hello!" is set to the property "WSNlabelString" of the instance. Then, the instance shows the string "Hello!" when the function executed.

2.10 Building of the application 'Hello' and executing it

It is better to save the created data of this project, at first. To save the project, select ((menu:Project → Save project)).

Then, the next operation to do is to build the application. Select ((menu:Build → Build all)), and the building dialog will appeared in order to confirm whether there are any compiling errors. If no error occurs, it has completed and the executable "hello" is available in the directory. By selecting ((menu:Build → Execute)), the application is executed.



[Building the application "Hello"]

Chapter 3

Application Window

3.1 What Is an Application Window?

An "application window" is a window which is created by the application builder. The application builder manages and saves each individual application window unit.

The followings show the different types of application windows created and managed by the application builder:

- Normal application window

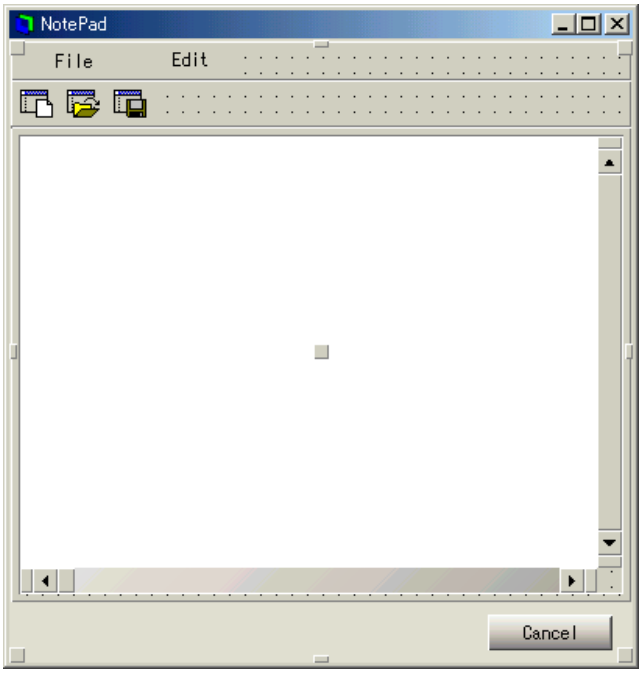
  This is the most commonly used application window. For this reason, we simply call it the "application window".

- Class application window

  This window creates a new C++ class by integrating classes and event procedures. The advantage of using this type of window is that it promotes efficiency of development.

- Stored application window

  This window is stored to an object file, where it can be called dynamically or even shared between multiple applications and processes. The advantage of this type of window is that it saves resources and allows reuse of a single design when writing several different modules or applications.

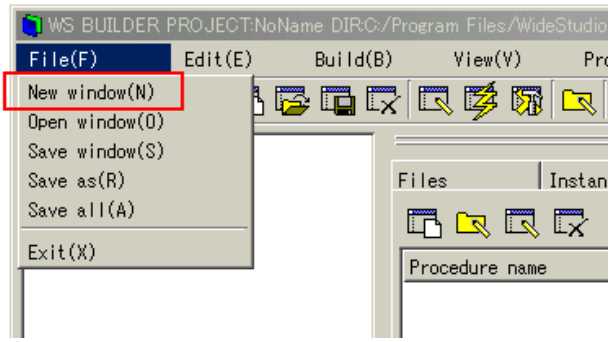


[An application window under construction...]

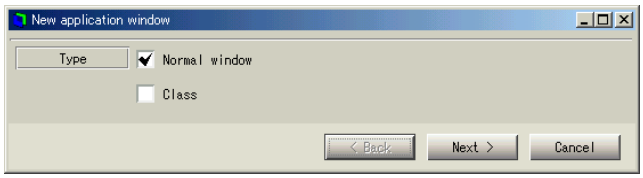
3.2 How to Create/Save a New Application Window

You can create a new application window by selecting ((menu:File → New Window)).

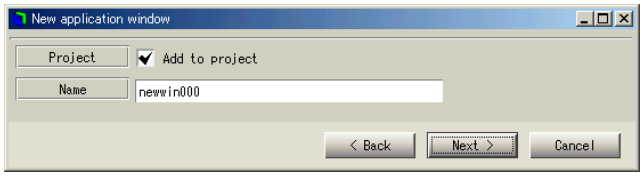
When the wizard dialog appears, type in the name of your new application window (default is newwin000), and then click "Finish" to create it. Your new application window is ready!



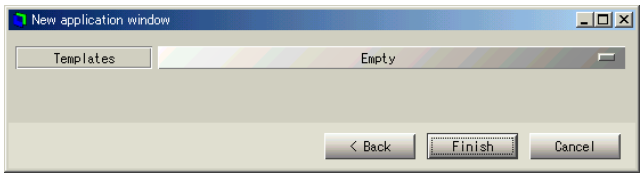
[Creating a new application window]



[The application window wizard (choice of type)]



[The application window wizard (fill the name of the window)]

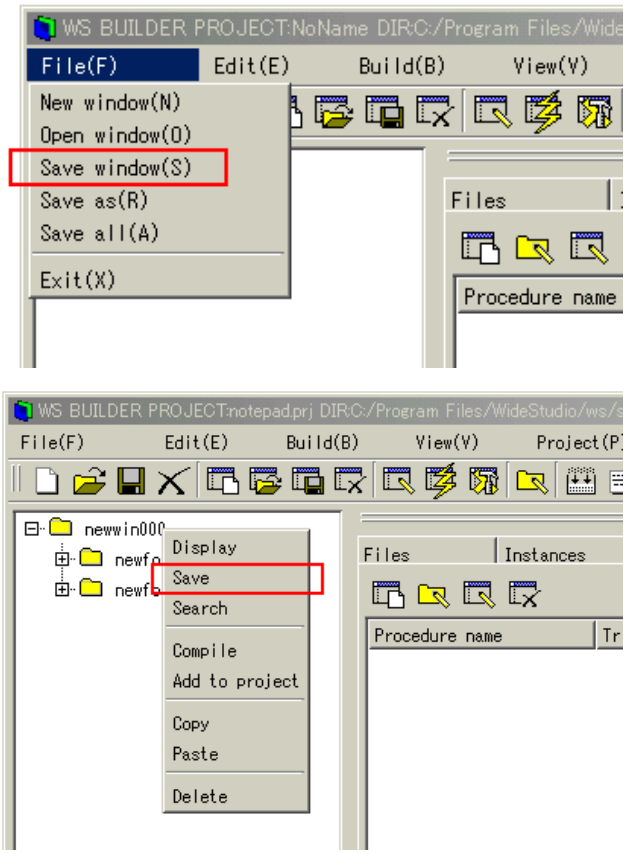


[The application window wizard (select one from the templates)]

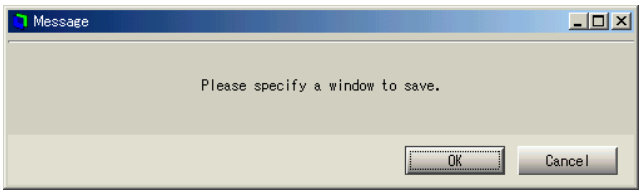
To save the application window, select it using the inspector. Then select ((menu:File → Save Window)). An alternate method of saving your window is by right-clicking on the inspector and selecting the "Save" option.

To save all application windows open currently, select ((menu:File → Save All))

The name of the saved file will have the form "[file name].win".

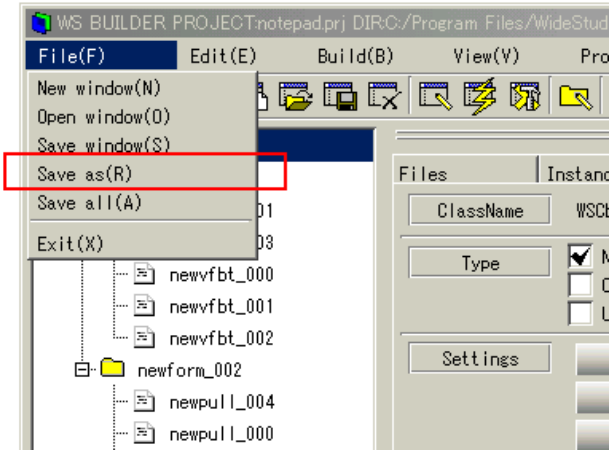


[Saving the application window]



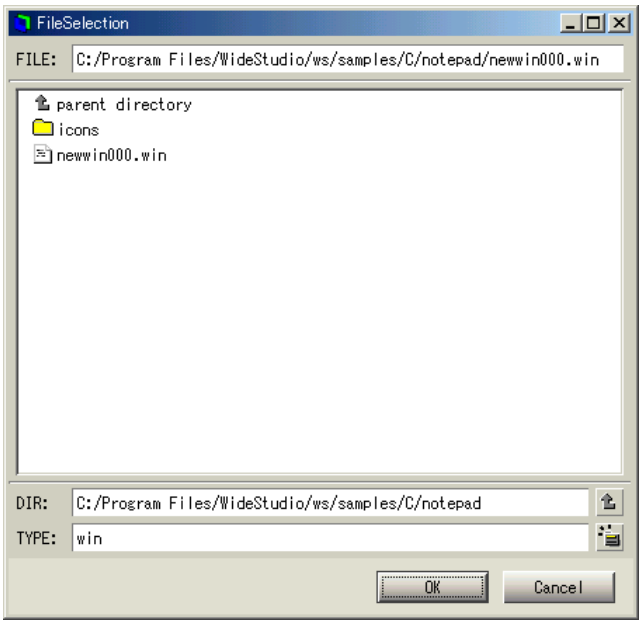
[A warning dialog]

To save the application window under a different name, select ((menu:File → Rename)).



[Renaming the application window]

The following is an example of how to rename a file:



[Input of a new file name]

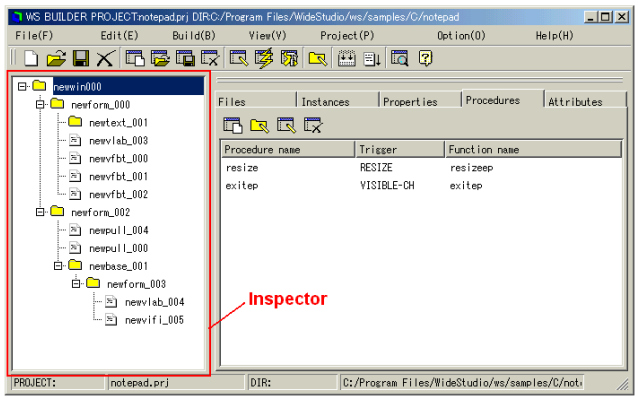
3.3 How To See The List of the Application Windows

3.3.1 How To See The List of the Application Windows

The inspector shows a list of the application windows and can controls each one.

It shows the information in a tree-style view so that you can easily recognize which instances are where and easily place them on the application window.

The top instance of the tree is an application window. For example, an instance: newwin000 in the following figure is an application window.

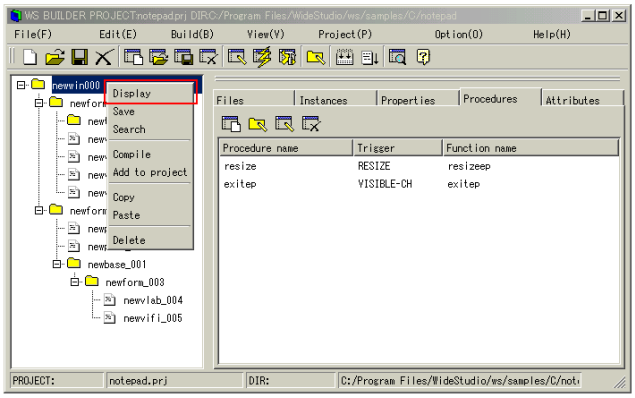


[The tree of instances of an application window]

3.3.2 How to display the application window to edit

To edit the application window, select it using the inspector, and choose ((menu:Edit → Display)). Alternatively, you can right-click on the item in the inspector and select [Display].

If you want to make the item invisible, select [Dismiss] from the pop-up menu for that item in the inspector.

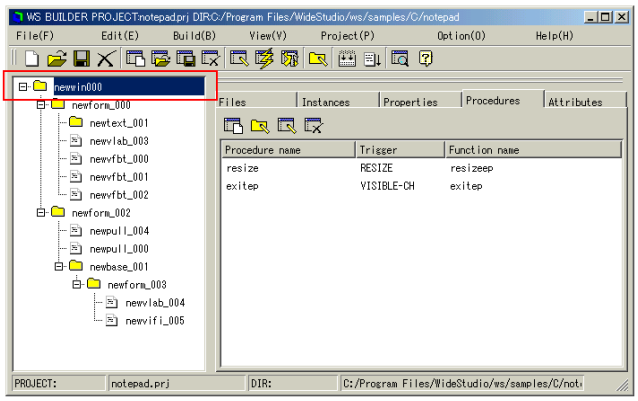


[Displays /Dismisses the application]

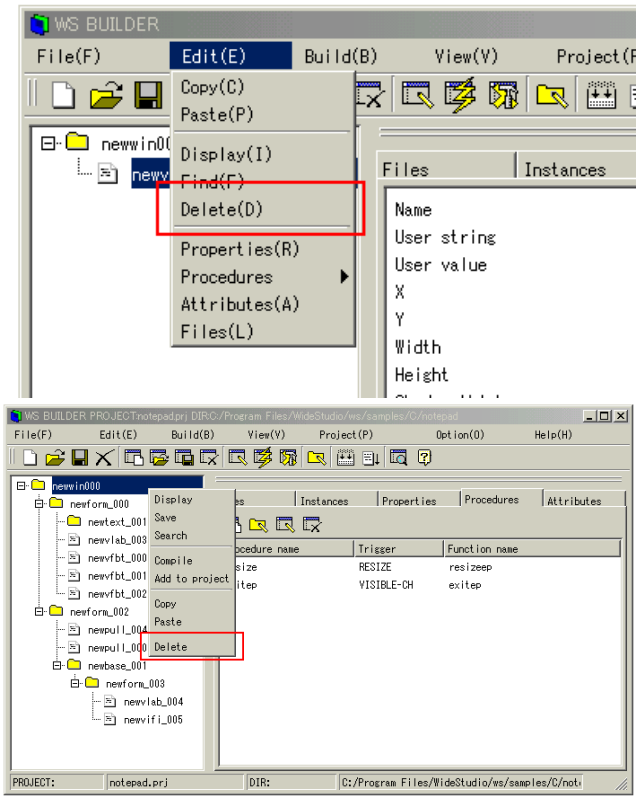
3.4 How to delete the application window

To delete the application window, select it using the inspector and then choose ((menu:Edit → Delete)) or [Delete] on the pop-up menu.

An example is shown below:



[Choosing the application window]

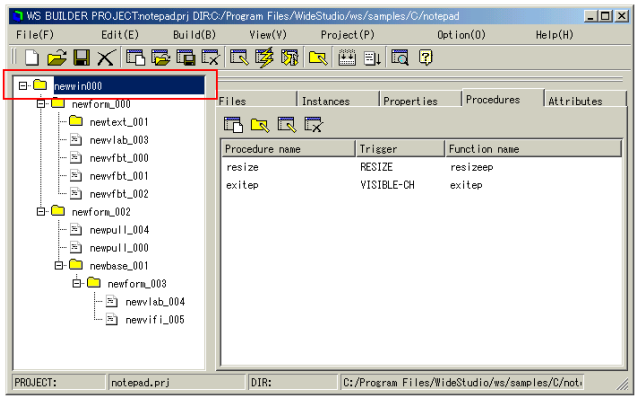


[Deleting the application window]

A dialog box will appear asking if you are sure you want to delete the instance. Select [OK] to delete it.

3.5 How to rename the application window

There are two ways to rename the application window: The first way is to change the WSNname property on the Property Editor, the other is to change it in the [Attributes] section of the inspector.

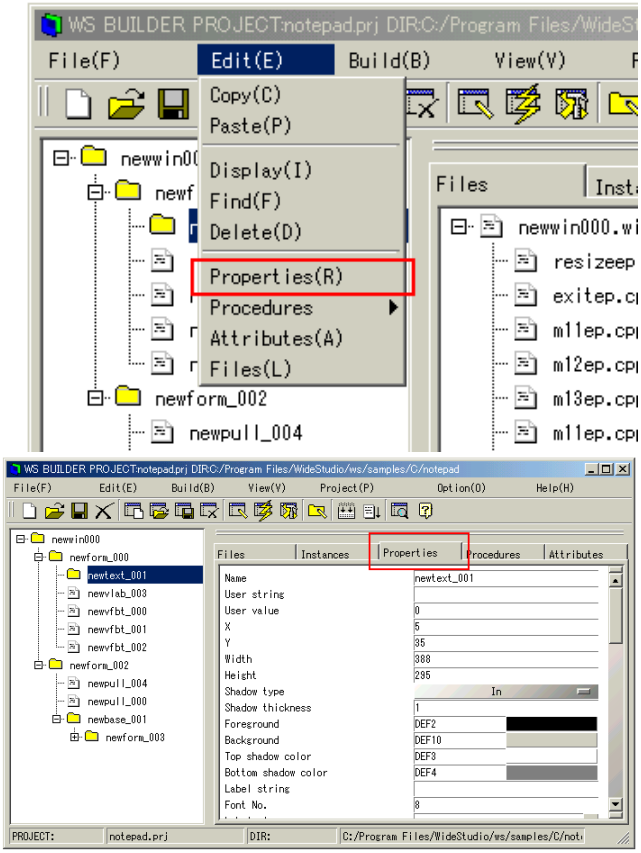


[Choosing an application window to rename]

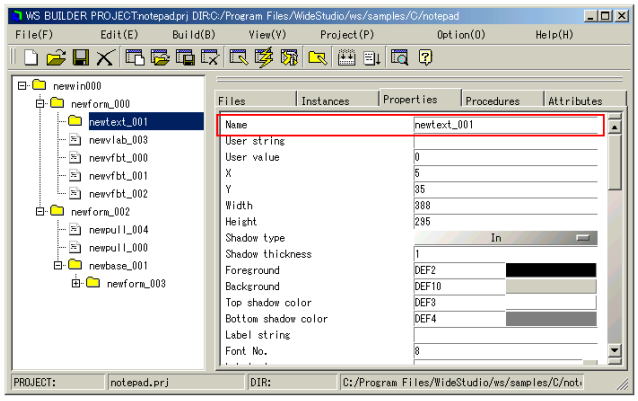
To change the name using the first method, select ((menu:Edit → Properties)), or select the [Properties] tab on the inspector.

Now find the WSNname property and change it's value.

(Notice) The name of an application window is the same of the instance of a top-level window.



[The properties]

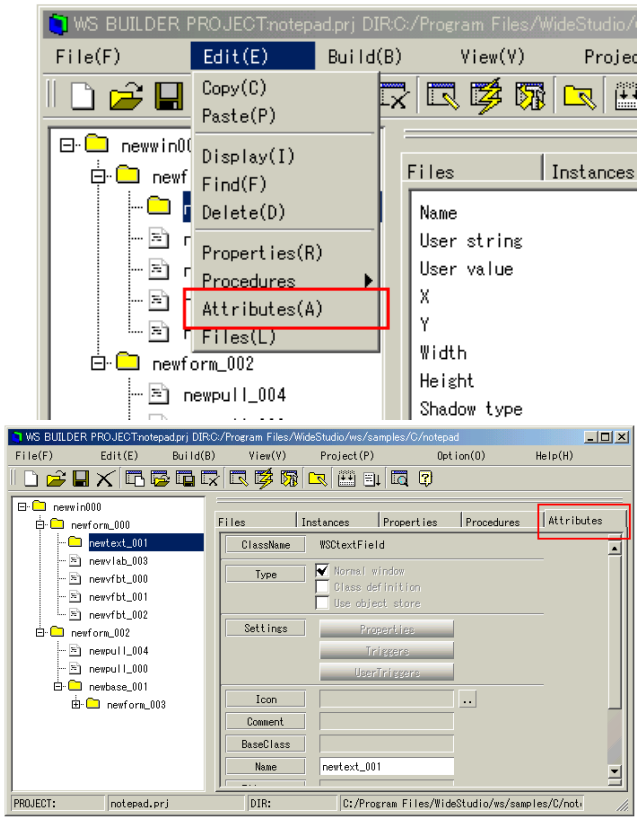


[Editing the name: WSNname property]

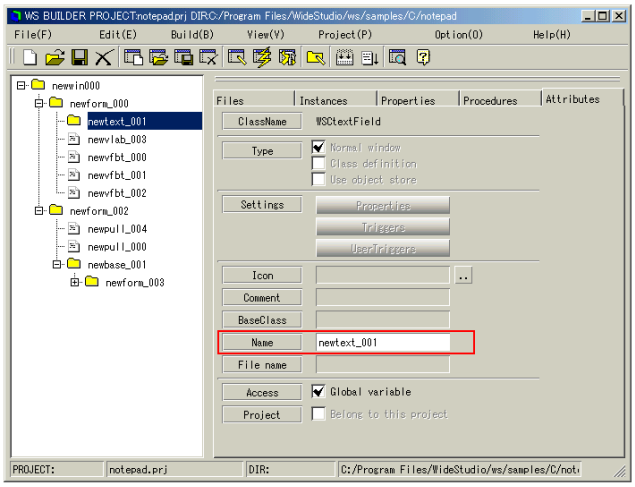
Using the second method, choose the [Attributes] tab on the inspector or choose ((menu:Edit → Attributes)).

Under the attributes window, put the new name into the [Name] field and press the [Update] button.

(Notice) This is the same as changing the WSNname property of the instance.



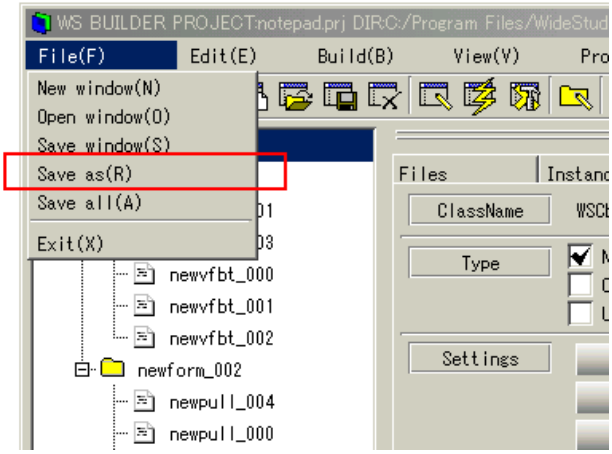
[The attributes]



[Changing the name]

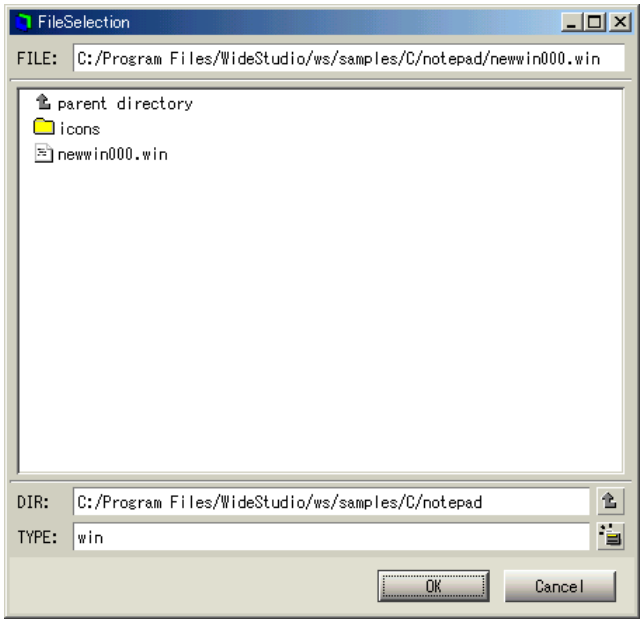
3.6 How to save the application window as specified file name

Usually, the file name of the application window is "[name].win". You can save it to a different file name by ((menu:File → Save as)).



[Save as]

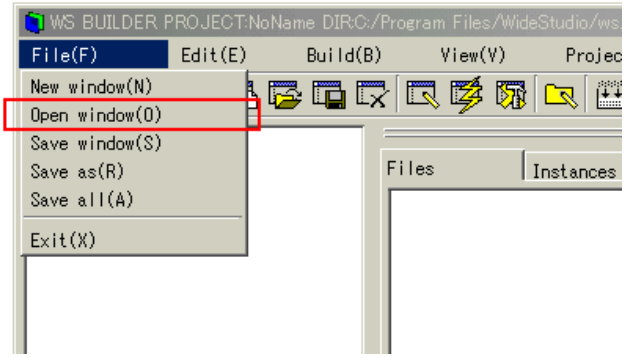
The following figure shows you how to save to a different file name.



[Input a new file name]

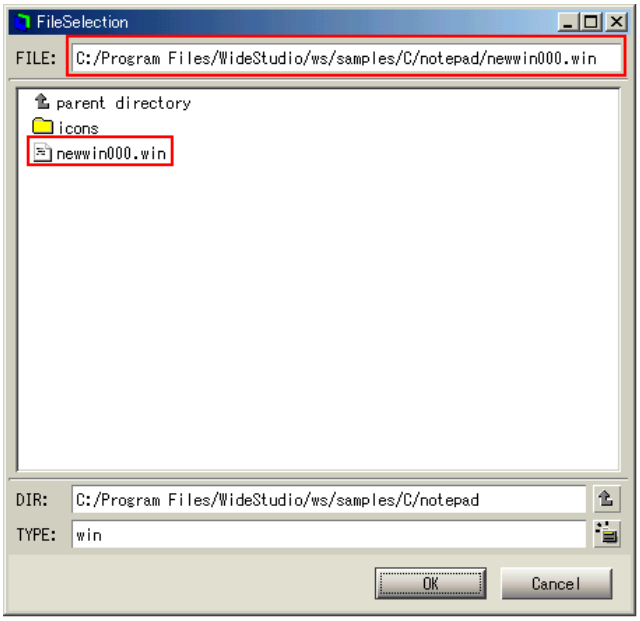
3.7 How to open the file of an application window

Select ((menu:File → Open window)) to load an application window from a file.



[Open an application window]

The following shows the file selection dialog to select a file of an application window.



[Selecting the file]

You can input the file name directly, or select by double-clicking the desired file name.

Chapter 4

Object

4.1 What are the GUI objects?

GUI objects like a window or push buttons are made by C++ classes and are easy to customize through the following functions.

- Property

  The GUI objects have properties - a particular kind of member variables, which the developer handles through an unified interface. There are properties like the color, width, shape, value, string, and so on.

- Event Procedure

  The developer sets the event procedures for each instance of a GUI object. Each procedure has a trigger and is executed when its trigger fires. The procedure handles it for that instance so you can add to the instances various kind of functionality which the objects do not have by definition. Therefore, you are free from creating many new classes which are unnecessary.

The following table shows the kinds of GUI objects which have the child management function.

	Class Name	Description
Windows	WSCwindow	Top-level window; base class for an application window
	WSCmainWindow	Top-level window; usually used
	WSCdialog	Base class for a dialog
	WSCmessageDialog	Dialog for popup a message
	WSCinputDialog	Dialog for input a text
	WSCwizardDialog	Dialog for switch panels like as a wizard software
	WSCfileSelect	window for file selection
	WSCworkingDialog	window with a prograss meter
Forms	WSCform	Rectangle area which can manage child objects
	WSCindexForm	Area which has tabs to switch sub areas
	WSCsform	Area with several sub areas resizable by separators
	WSCscrForm	Scroll area
	WSCradioGroup	Groups radio buttons
	WSCcheckGroup	Groups checkboxes (radio buttons)
	WSCvertForm	Form for vertical alignment
	WSChorzForm	Form for horizontal alignment
	WSCmenuArea	Bar for pull-down menu
	WSCffrom	Floating form that can be an independent window
	WSCprform	Form that can be printed and exported as PostScript
	WSCj3wform	Form for J3W; 3D graphic library
	WSCopenglform	Form for OpenGL

The following table shows kinds of the objects which is placed as a child; do not have the child management.

	Class Name	Description
Label / Button	WSCvbtn	Push button
	WSCvtoggle	Toggle button
	WSCvradio	Checkbox (radio button)
	WSCvlabel	Text label with border
	WSCvklabel	Label with keyboard focus
	WSCvslabel	Label with string selection for copy & past
	WSCvarrow	Button with triangle arrow
Control	WSCpulldownMenu	Pull-down menu
	WSCpopupMenu	Pop-up menu
	WSCoption	Option menu to choose values
	WSCcomboBox	Input field combined with option menu
	WSCvscrBar	Scroll bar to adjust a value of the bar
	WSCvslider	Slider to adjust a value of the bar
	WSCvmetger	Prograss meter
	WSCvclock	Digital clock
	WSCvifield	Text input field
	WSCvpifield	Text input field with mask for password input
	WSCvmifield	Multi-line text input field
	WSCtextField	Scrolled text input field
	WSClist	List of items
	WSCtreeList	Tree list
	WSCdirLTree	Tree of file system directory
	WSCverbList	List of strings
	WSCgrid	Grid with cells that can be input directly
	WSCvimage	Label for displaying image

	Class Name	Description
Drawing	WSCdrawingArea	Area for drawing figures
	WSCvarc	Arc or circle
	WSCvrect	Rectangle
	WSCvline	Line
	WSCvpoly	Polygon
	WSCvlineGraph	Line graph
	WSCvbarGraph	Bar graph
	WSCvgraphMatrix	Scale grid for graph
	WSCvgraphScale	Horizontal scale for graph
NonGUI	WSCngbase	Base class for nonGUI classes
	WSCvtimer	Timer
	WSCvspace	Space that fills spaces between objects
	WSCvballoonHelp	Popup balloon help
	WSCvcsocket	Client socket for TCP communication
	WSCvssocket	Server socket for TCP communication
	WSCvudpsocket	Socket for UDP communication
	WSCvremoteClient	Client of remote instance
	WSCvremoteServer	Server of remote instance
	WSCvdb	Database client of SQL
	WSCvodbc	Database client of ODBC SQL

4.2 Windows

The top-level windows can be categorized roughly into two parts: Windows for general use and Dialogs for prompting users to input something.

Usually, the WSCmainWindows class is used for a top-level window, but you can set any class name, such as Dialog, to be the default base for a top-level window in the project settings.

- WSCmainWindow

  Used for a general top-level window. An instance of this class is created by default when a new application window is created.

- WSCdialog

  Base window for a Dialog. A Dialog is equipped with "OK" and "NO" buttons beforehand, so you can get these buttons shown and this window focused when needed.

- WSCmessageDialog

  Dialog for displaying messages. Shows messages to get users noticed.

- WSCfileSelect

  Dialog for file selection. Use this to give users the choice of a file or directory.

- WSCwizardDialog

  Dialog with switching screen functions by the "¡ Back" "Next ¿" buttons.

4.3 Forms

The Forms manages the client region of the parent window by allowing other objects to be placed on it. Objects on a Form move along as the Form moves. Also, when a Form is displayed, the objects on it appear too.

- WSCform

  Used for a general Form. This Form provides coordinates dependent from the parent window, and raises a resize event. When a resize event is raised, an event procedure for that event allows for complex replacing which is difficult for users to do by only setting anchor properties.

- WSCscrForm

  Form with scrolling function. Useful for handling an area larger than the actual window size. Use Scroll bar to scroll the area. You can set various properties such as the size of the area, the scrolling unit and so on.

- WSCsform

  Form with a separator to separate regions. This separator has a mouse scaling to decide the size of the area.

- WSCindexForm

  Form with indexing tab. Select an index to show the corresponding area. This form appears as if it has multiple layers.

4.4 Commands

Command controls are used chiefly for displaying, to allow users to raise events by using the mouse (Push buttons), to select a value (Selecting value), to select an action out of a menu (Pull-down menu) or to input/edit strings by keyboard (Pull-down menu).

- DISPLAYING

  WSCvlabel (Strings, Pictures), WSCvclock (Clock), WSCvmeter (Meter)

- PUSHING BUTTON

  WSCvbtn, WSCvfbtn

  Clicking the mouse on it raises an ACTIVATE event.

- INPUT VIA KEYBOARD

  WSCvifield, WSCvmifield, WSCtextField

  All of them are used to input texts.

- SELECTING VALUE

  WSCvradio, WSCoption, WSCcomboBox, WSClist, WSCvslider

  Used for selecting a value.

- PULL-DOWN MENUS

  WSCpulldownMenu, WSCpopupMenu

  An event procedure set on the menu item activates when the item is selected.

4.5 Drawings

The Drawing controls represent various kinds of graphics.

- Circle, Oval, Arc, Filled Circle

  WSCvarc

- Line, Polygon

  WSCvline

- Rectangle, Filled rectangle

  WSCvrect

- Polygon, Filled polygon

  WSCvpoly

- Free graphics

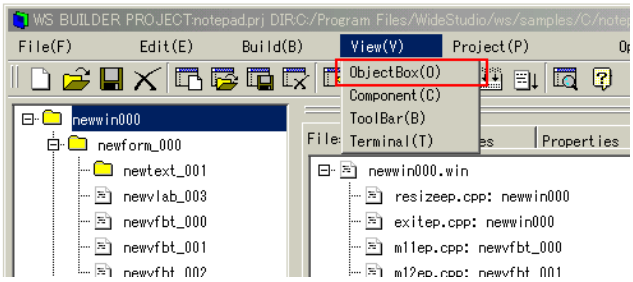
  WSCvdrawingArea

- Graph

  WSCvbarGraph (the bar graph), WSCvlineGraph (the line graph), WSCvgraphScale (the scale of the graph), WSCvgraphMatrix (the lattice of the graph)

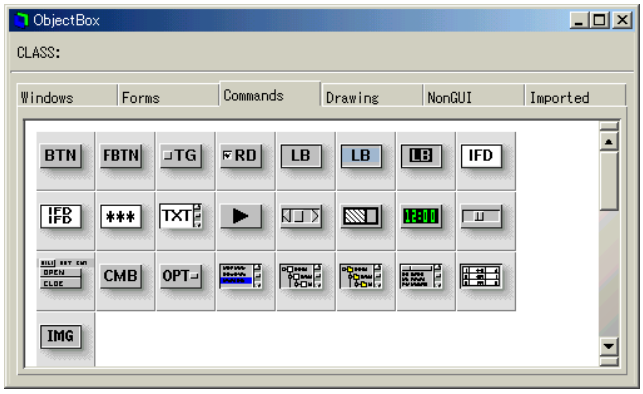
4.6 Place instances on an application window

The dialog for creating an instance is the object dialog box. Select ((menu:View → ObjectBox)) to pop it up.



[To pop-up the object dialog box]

The following figure shows the object dialog box.



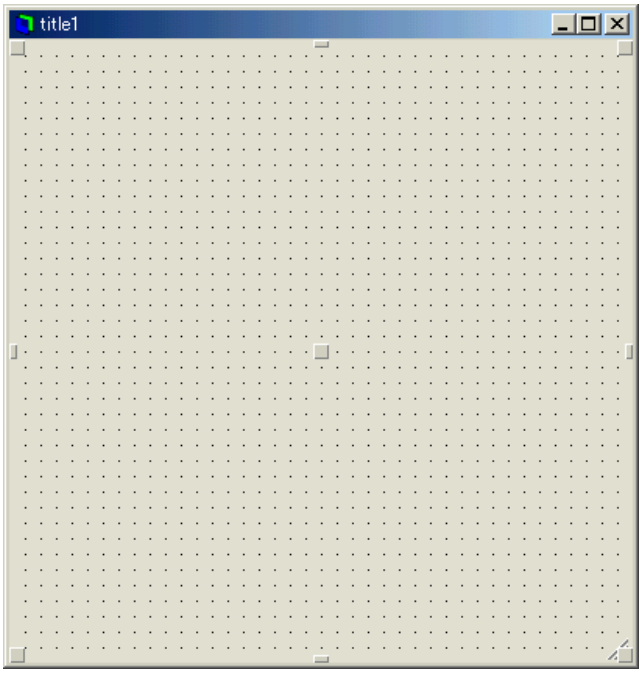
[A view of the object box dialog]

If you move the mouse pointer over the icons in the object dialog box, it shows its class name and some description in a balloon help. You can drag and drop an icon to the instance of the application window which is displayed.

See the section:[How to see the list of the application windows]. After displaying the application window, drag the icon and drop it on the instance to place the object.

The following shows the kinds of objects.

Kind	Description
Windows	Top windows or dialogs which receive a border from the window manager
Forms	Areas which have window resources and manage child objects
Commands	Objects which have various functions, and do not have window resources
Imported	Objects which are imported by the external libraries



[A new push button on the application window.]

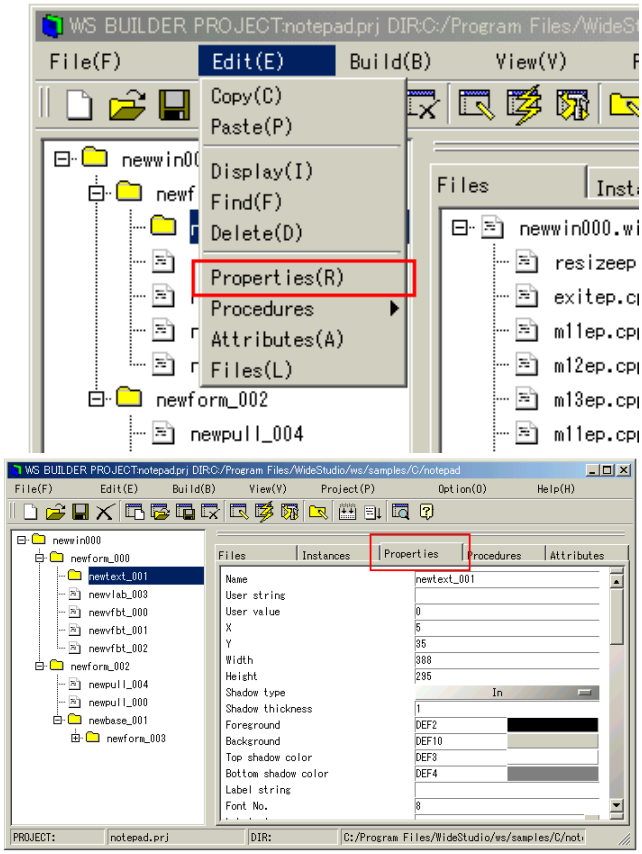
We call the object which manages child instances "Manager". The following shows manager objects.

Manager	Class Name	Description
Windows	WSCwindow	Top window which is the base of the application window
	WSCdialog	Dialog
Forms	WSCform	Area which has window resources.
	WSCscrForm	Scrolled area
	WSCsform	Area which has several sub-areas and separators
	WSCindexForm	Area which has some index tabs

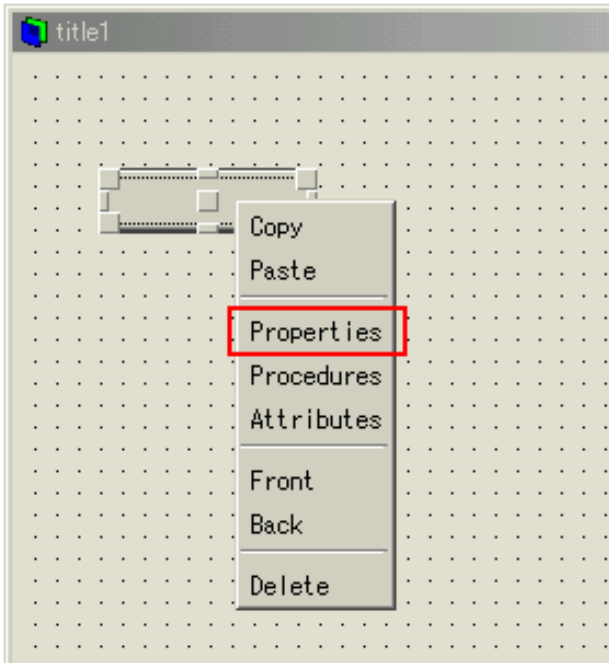
The instances of manager objects show a grid made of dots in editing mode on the application builder.

4.7 Set Properties

Each GUI object has internal variables, called "properties", which define its shape, color, and actions. To customize an instance, you alter its properties. The properties are set via the Property Editor of the Inspector. The Property Editor can be opened in two ways: ((Builder → menu:Edit → Properties)) or ((Inspector → tab:Properties)).

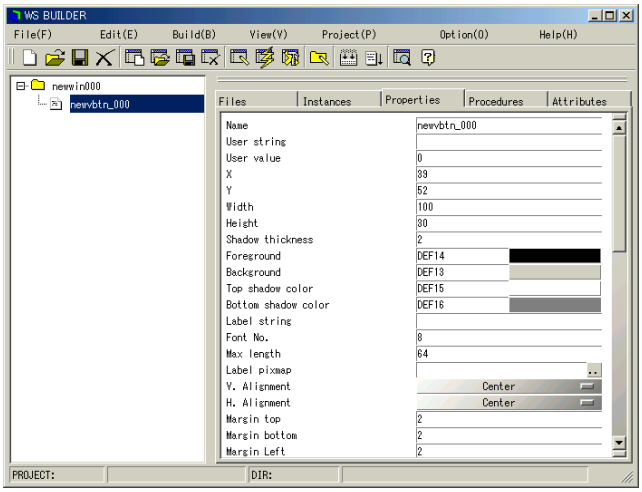


[Above, how to access Properties from (top) the Builder's menu-bar and (bottom) the Inspector.]



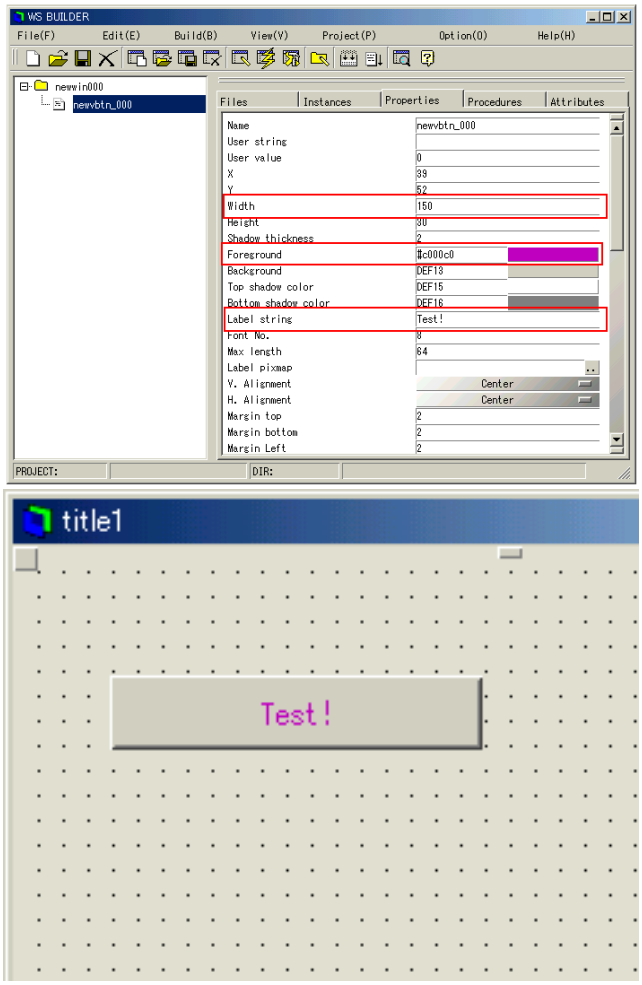
[Above, how to access Properties from an Instance's context-menu.]

Using any of the above three methods, you'll get a properties sheet similar to the one below. This example shows the properties of WSCvbtn (a push button).



[Above, view of the Properties sheet (to the right of the Inspector).]

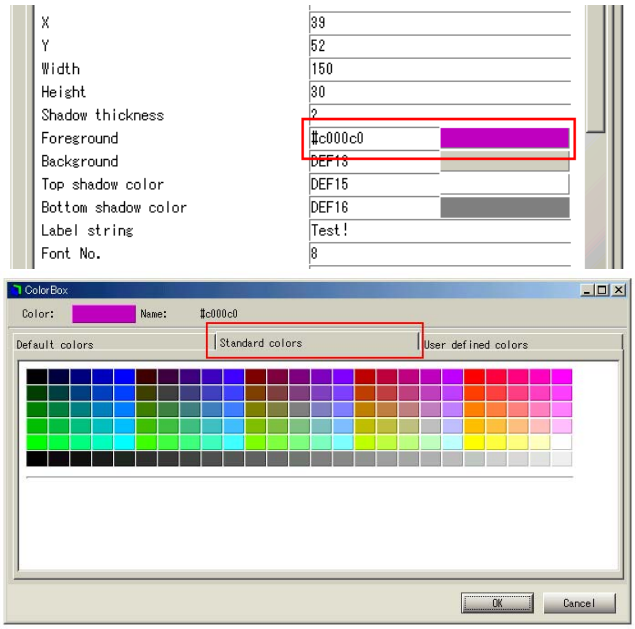
The properties sheet contains many rows, one per property. Each row has a label, to the left, and a control (most often a text box), to the right. After editing a property, hit [Enter] to confirm the change, and send it to the instance. The example below shows three properties being changed ("Width", "Foreground", and "Label string"), and their effects upon the instance.



[Above, (top) changes in the Properties sheet, and (below) corresponding changes in the Instance.]

You can specify a hexadecimal RGB value, or press color button to display the color selection dialog, when the property is a color property.

Color properties ("Foreground", "Background", "Top shadow color", "Bottom shadow color") can be adjusted in one of two ways. You can either click the color swatch to open a special color selection dialog box, or directly type a six-digit hexadecimal RGB value.



[Above, setting the "Foreground" color property to purple. Top, the property in the Properties sheet. Bottom, the color selection dialog.]

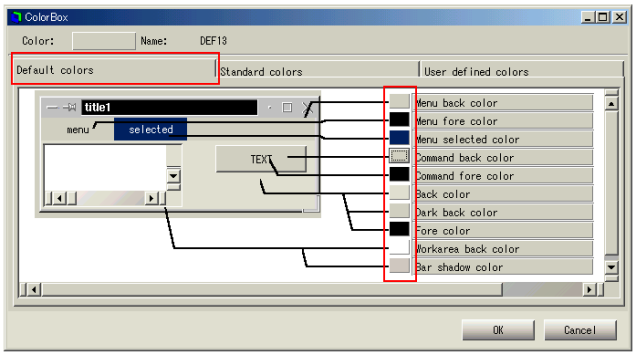
A hexadecimal color value is represented by a "number" sign ("#") followed by three hexadecimal pairs, representing the intensity of Red, Green, and Blue. A hex digit is 0-9,a,b,c,d,e,f. "00" is zero intensity, and "ff" is maximum. Hence, white is "#ffffff", medium grey is "#999999", and black is "#000000". Red is "#ff0000", medium green is "#00cc00", dark blue is "#000033", and so forth. Valid colors are in the range:

#00000 - #fffff

(Most computer graphics applications represent colors in this way.)

You can also specify system default colors with the color selection dialog. You have better use these colors on development for Windows system.

The color selection dialog permits you to use the standard Windows system colors, that is, the ones shared by all the applications. When developing a program for use under Windows, use these to ensure that it matches everybody else.

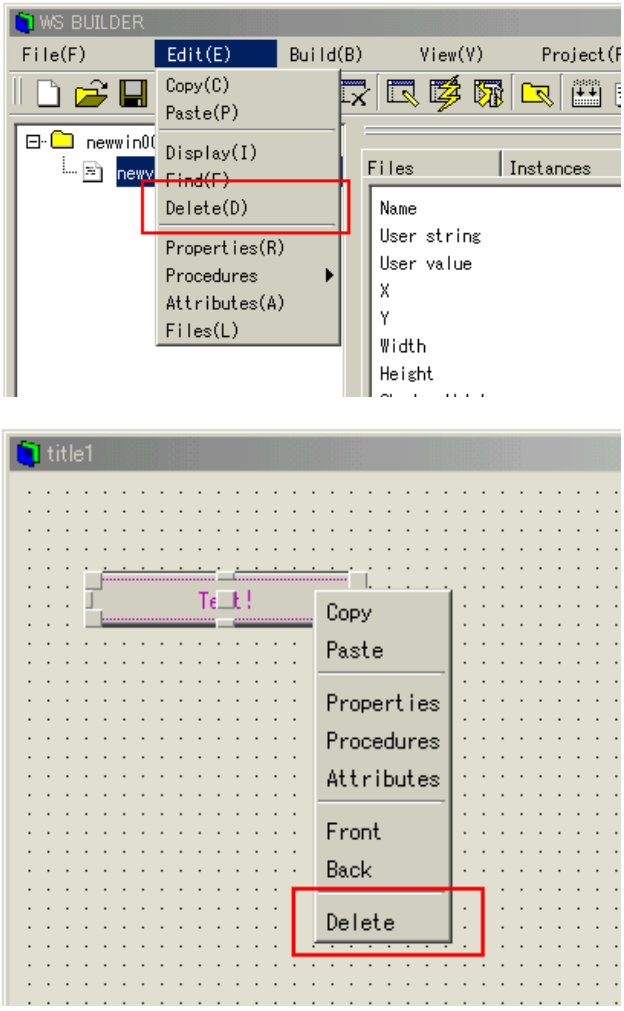


[Above: how to select the system default colors.]

4.8 Delete an instance and a window

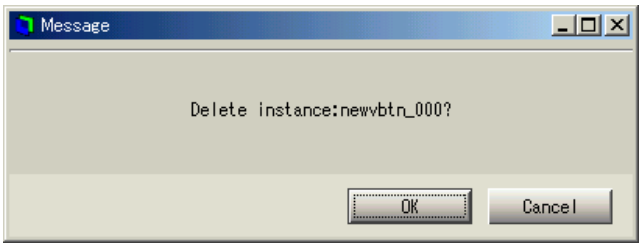
4.8.1 How to delete the instance

To delete an Instance, first select it, then issue one of these two commands: ((menu:Edit → Delete)) or ((context menu → Delete))



[Above: deleting an Instance by (a) the menu-bar or (b) the Instance's context menu.]

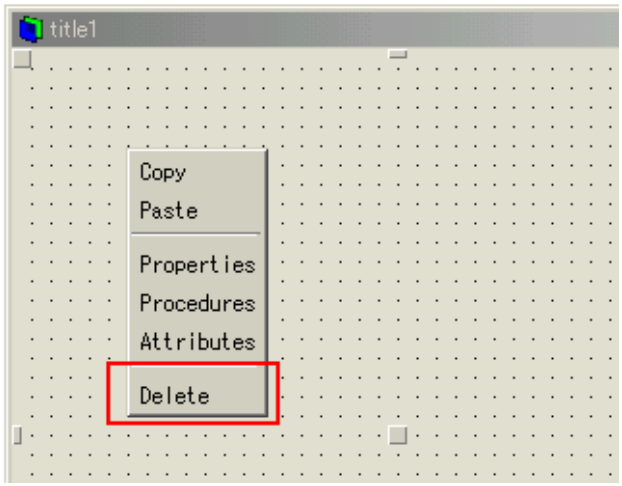
A confirmation dialog will appear. Click [OK] to delete the instance – and also, all of its child instances are deleted.



[Above: delete confirmation dialog]

4.8.2 How to delete the application window

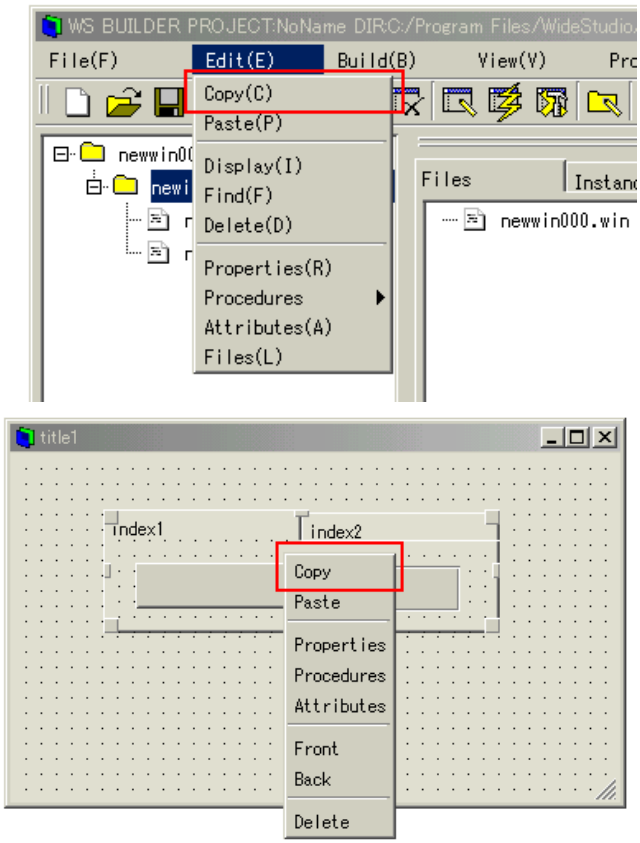
As an alternative to the method described in Chapter 1, you can delete the application window using its context menu, in the same way you would delete an Instance. Just right-click on a blank spot in the application window to invoke the context menu, and select [Delete].



[Above: deleting the application window using the context menu.]

4.9 Copy instance(s)

First, select the Instance you wish to copy. Either select it in the Inspector's list, or in the application window. Next, invoke the copy command. As usual, there are two ways to do this: ((Builder → menu:Edit → Copy)) or ((context menu → Copy))



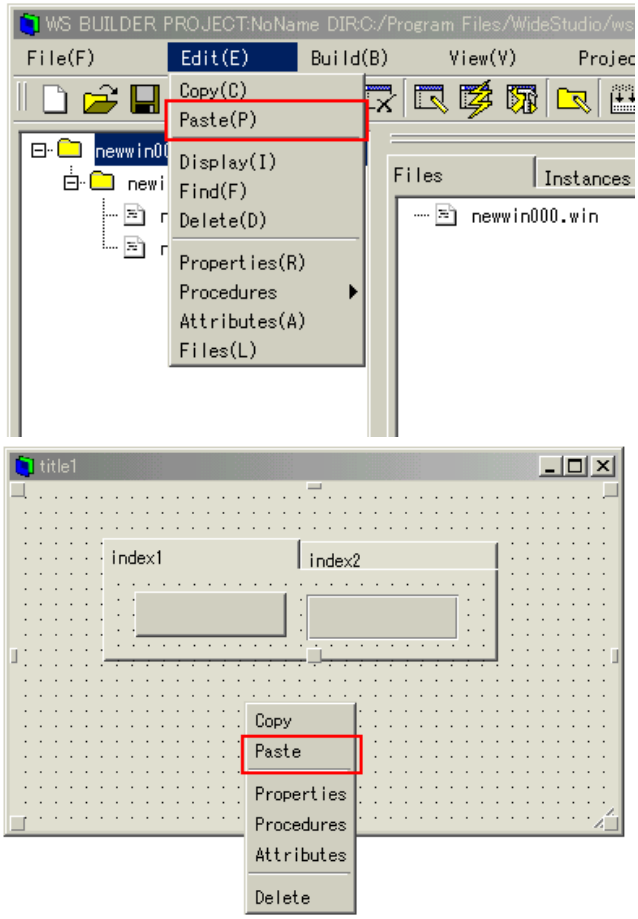
[Above, copying an Instance using (top) the Builder's menu-bar and (bottom) the context menu.]

Select the instance to paste with the inspector or by clicking directly, next. then you can paste the instances on it, with [Paste] of the [Edit] menu or [Paste] of the right button pop-up menu.

Select the instance either by the Inspector, or by clicking on it.

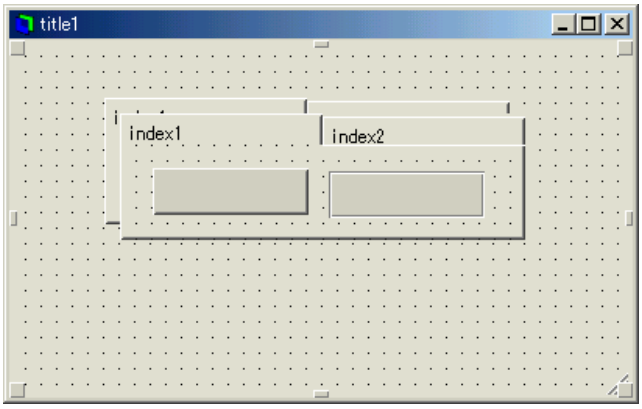
To paste, execute either of the following commands: ((menu-bar → Edit → Paste)) or ((context → Paste)).

To paste the copy into existing object (either another Instance, or the application window itself), first select the object (either in the Inspector's list, or in the application window). Next, invoke the paste command, either: ((Builder → menu:Edit → Paste)) or ((context menu → Paste)).



[Above, pasting a copied Instance using (top) the Builder's menu-bar and (bottom) the context menu.]

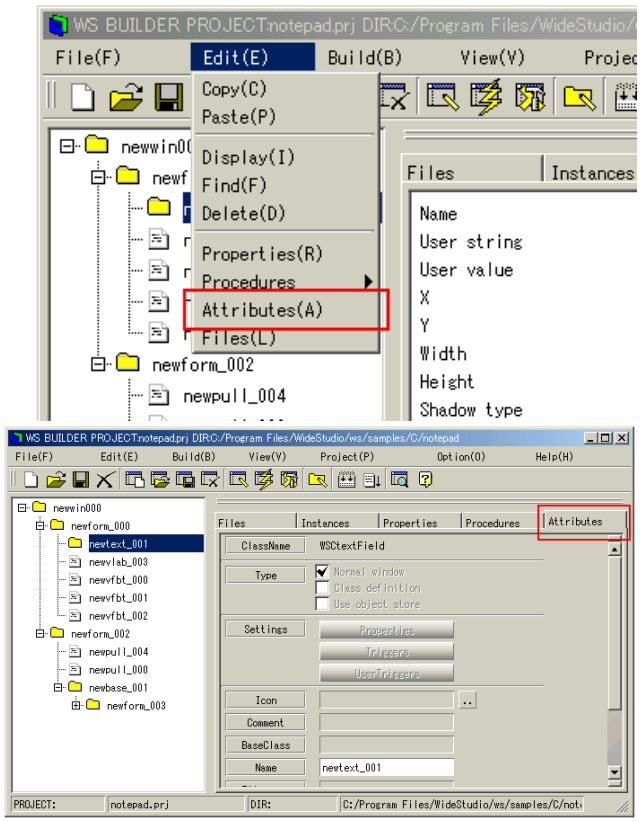
The following screen shot shows the copy, having been pasted atop the original.



[Above, an Instance pasted to the application window.]

4.10 Make an instance be an external variable

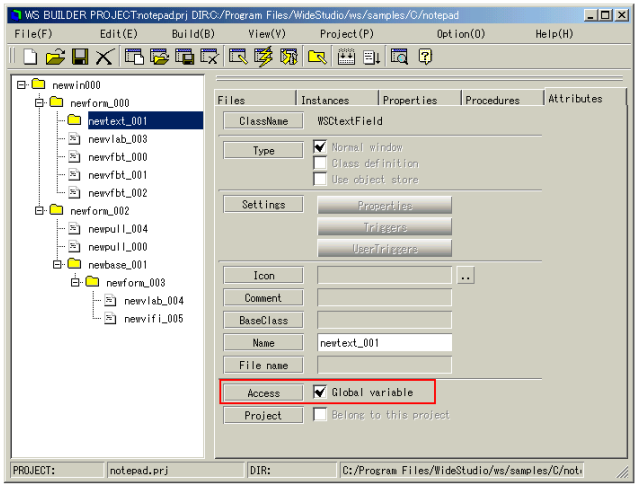
You can make an Instance be an "external" variable, so that it can be accessed and manipulated by a C++ program. First, open the Attributes tab of the Inspector, either by: ((Builder menu:Edit → Attributes)) or ((Inspector → tab:Attributes)).



[Above, activating the Attributes tab of the Inspector.]

Select the "Access / Global variable" check-box.

Note: the application window is always an external variable: you don't need to set it, and you can't change it. Instances defined as Arrays are also automatically external.

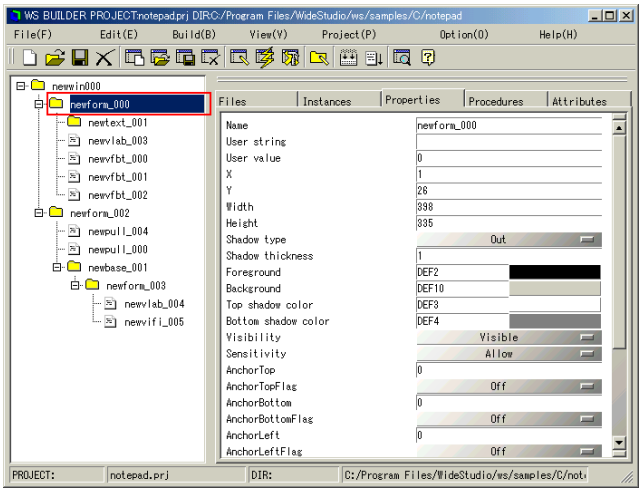


[Make the instance a global/external variable]

4.11 View the list of instances

4.11.1 How to see a composition of the instances of the application window

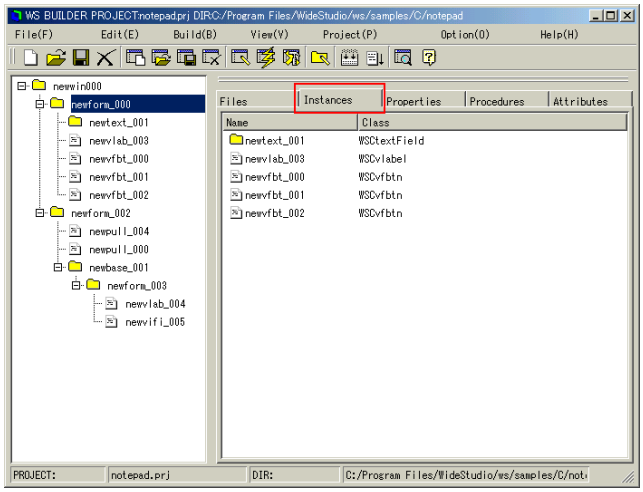
You can display a tree which shows the composition of the instances by clicking where the following figure indicates, or by double-clicking the instance name. It shows the child instance of the selected instance, and closes them once again.



[A composition of the instances]

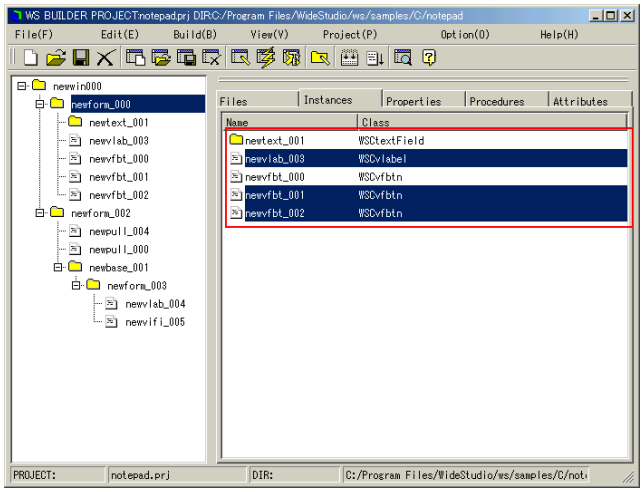
4.11.2 How to see the details of the children

You can see the details of the children by clicking the [Instances] tab of the inspector.



[The details of the children]

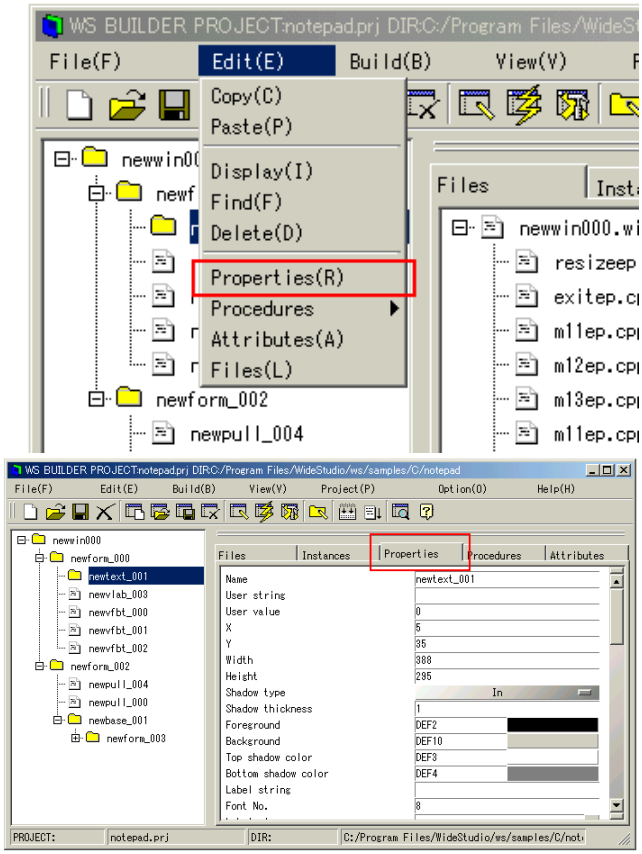
You can copy the children by multi-selection within the Detail List. Select the instances as follows, and execute [Copy] and [Paste] from the [Edit] menu.



[A multi-selection of the instances]

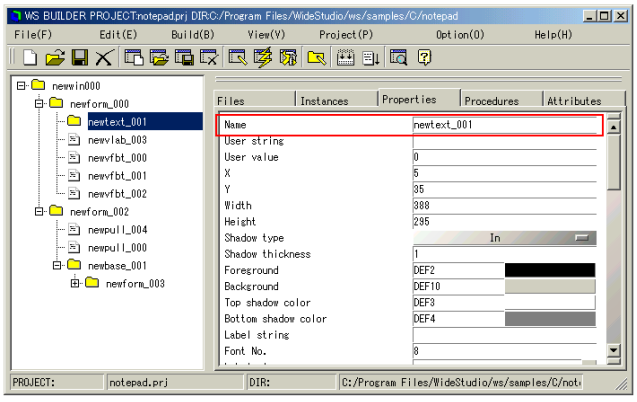
4.12 Rename the name of an instance

The application builder names the instances when they are placed on the application window. You can change the name with the Property Settings of that instance. Select ((menu:Edit → Properties)) or the [Properties] tab of the inspector.



[The properties menu]

You can change the instance's name with the WSNname Property. In addition, when the instance is the top window of the application window, it reflects the application's window name. See the section:[How to rename the application window].



[WSNname Property of the instance]

4.13 Define Instances as an Array

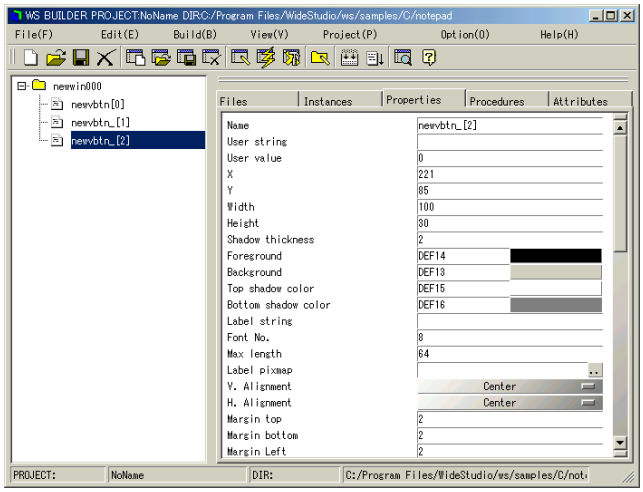
You can make the instances become an array, by giving them an array name. See the section:[How to rename the instances] to rename them.

Specify the array name as follows.

arrayname[No]

"arrayname" is given by alphabet and number which is used for an array name, "No" is the index number of the element and must begin from 0.

You can only make the instances become an array if they are the same class type, and exist on the same application window. So you can not make the application windows become an array.



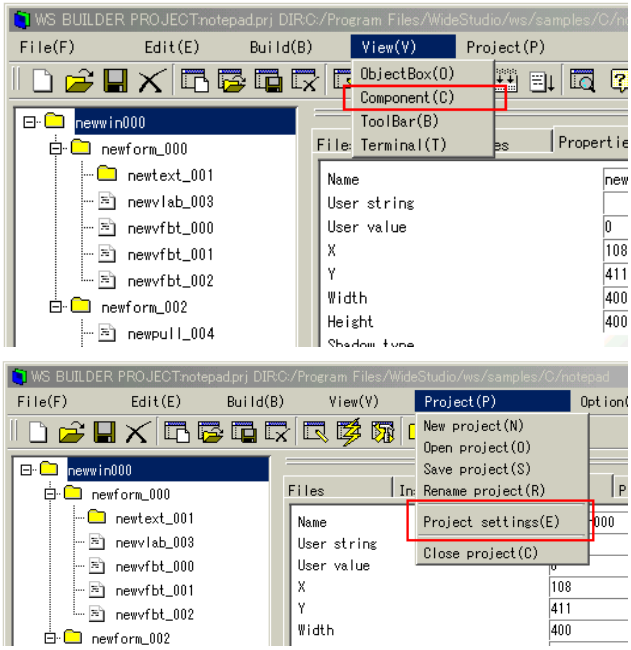
[The arrayed instance.]

See the section:[How to access to the arrayed instance] of the Programming Guide.

4.14 Use new classes imported fron a class libraries

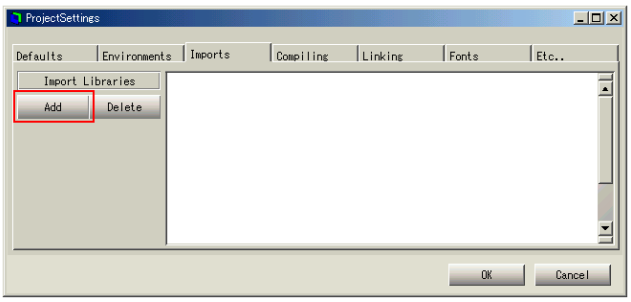
You can use the new classes of import libraries on the application builder. Specify import libraries with the Project Setting dialog:[Imports] section, which pops up by selecting [Component] from the [View] menu or [Project settings] from the [Project] menu.

As you add them, you should specify the linking libraries on the [Linking] section of the dialog as well. See section: [How to specify libraries for linkage] for detail.

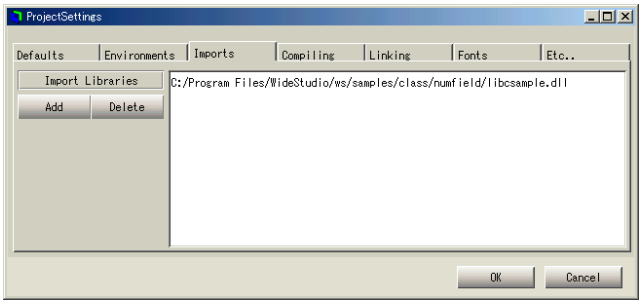


[View of the project setting dialog]

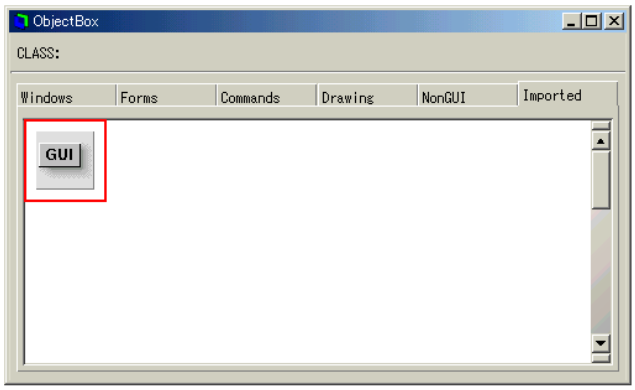
Push the [Add] button to add a import library, and specify it on the File Selection dialog which pops up. The import library is a dynamic link library for WideStudio.



[Importing a library]

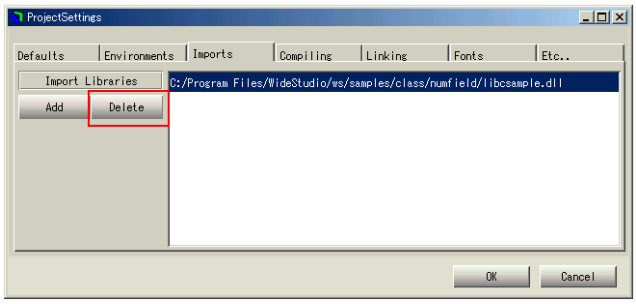


[A library imported]



[A class which is in the imported library]

If you want change the imported library, select it on the Project Setting dialog, as the following figure shows. And press the [Delete] button to delete it, then add the new library you want.



[Deleting the imported library]

4.15 See the class name of an instance

Selecting ((menu:Edit → Properties)) will show the class name of the placed object. It displays the class name in the class name form.

Chapter 5

Event Procedure

5.1 What is the Event procedure?

The event procedure is used to do something when an event is fired, in order to do a complicated task which is too difficult to realize within the object. You can code an event procedure with usual knowledge of the C++ language.

It has a trigger and a client instance, and is executed when the trigger is fired on the client. See the Programming Guide. The following shows what the event procedure has.

- Name

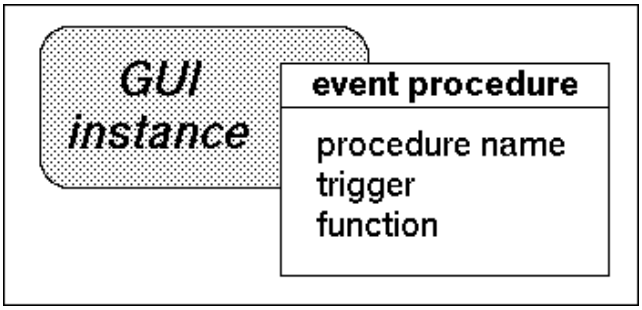
  An instance has many event procedures, so they have their names to identify them.

- C++ function

  The event procedure has a function which is coded in the c++ language. The function is executed if the event is fired.

- Trigger

  The event procedure waits for the event on the client instance. If the event is fired, it executes its function.



[The event procedure]

5.1.1 The function for the event procedures

The function has one parameter in which a pointer to the client object is passed. The following function is a template generated by the application.

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

//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void sample(WSCbase* object){
  //do something...

}
static WSCfunctionRegister  op("sample",(void*)sample);
```

5.2 Triggers

There are four kinds of triggers as follows.

- Change in the state of the instance

  It is fired by changing a state of the instance.

- Mouse pointer event

  It is fired by the mouse being clicked, moved, and so on.

- Keyboard event

  It is fired by the keyboard.

- The others

Change of the state	Description
WSEV_INITIALIZE	instance is initialized.
WSEV_DELETE	instance is released.
WSEV_ACTIVATE	instance is activated.
WSEV_VALUE_CH	value of the instance changes.
WSEV_VISIBLE_CH	visibility of the instance changes.
WSEV_PARENT_VISIBLE_CH	visibility of the parent instance changes.
WSEV_EXPOSE	instance is exposed.
WSEV_RESIZE	instance is resized.
WSEV_SENSITIVE_CH	sensitivity of the instance changes.
WSEV_PARENT_SENSITIVE_CH	sensitivity of the parent instance changes.

About the mouse pointer	Description
WSEV_MOUSE_IN	the mouse pointer comes into the area.
WSEV_MOUSE_OUT	the mouse pointer goes away from the area.
WSEV_MOUSE_PRESS	a mouse button is pressed.
WSEV_MOUSE_RELEASE	a mouse button is released.
WSEV_MOUSE_MOVE	the mouse pointer is moved over the area

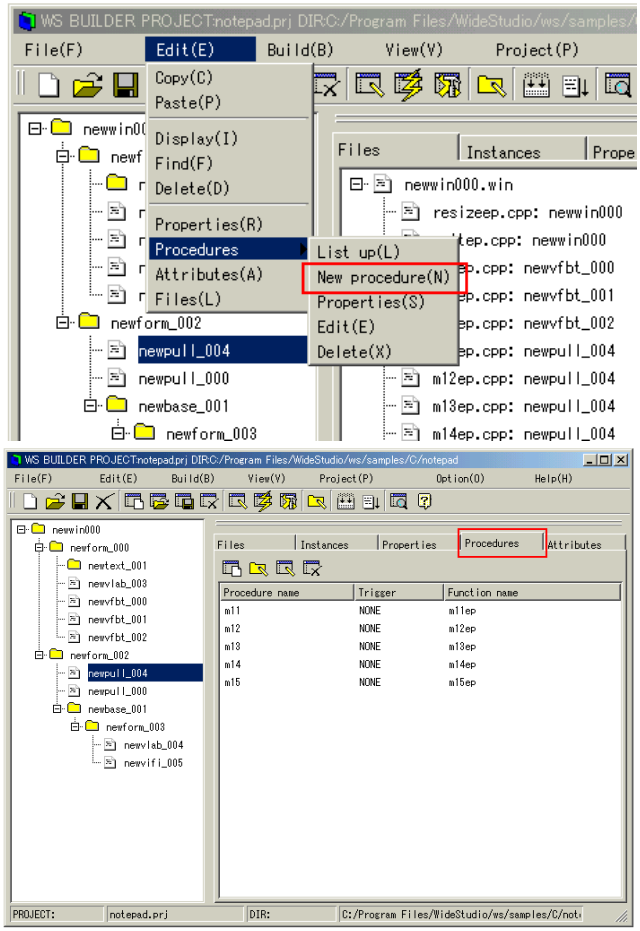
About the keyboard	Description
WSEV_FOCUS_CH	the focus changes.
WSEV_KEY_PRESS	a key is pressed.
WSEV_KEY_RELEASE	a key is released.
WSEV_KEY_HOOK	Hooking of the key event before dispatching.

The other	Description
WSEV_NONE	None.

5.3 How to create / setup / delete event procedures

5.3.1 How to see the event procedures of the instance

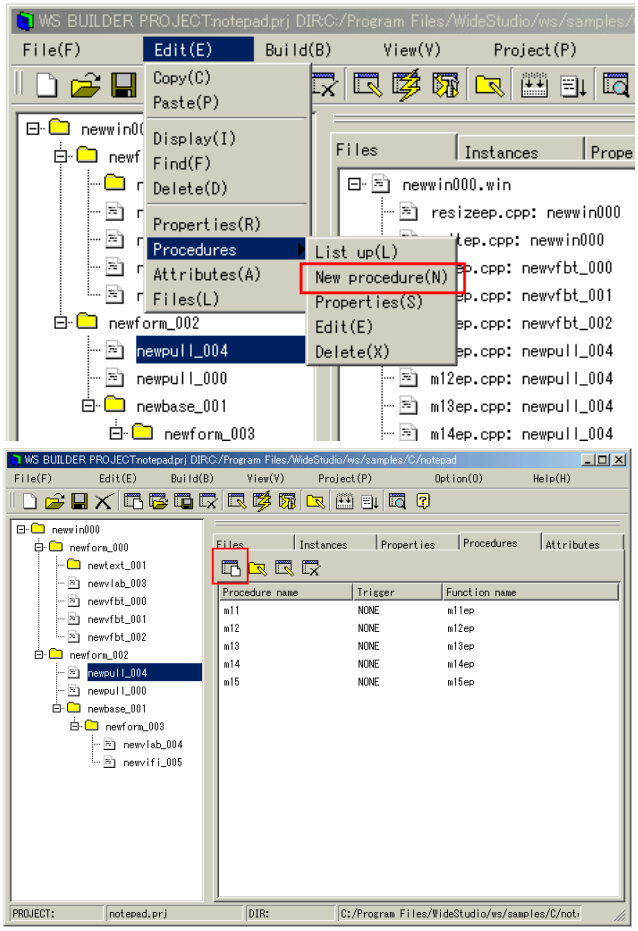
It is the procedure window of the inspector which is used for creating the event procedures. Select ((menu:Edit → Procedures → List up)) or the [Procedures] tab of the inspector to display the procedure window.



[A view of the procedure window]

5.3.2 How to create an event procedure

Select the target instance for the event procedure on the inspector, then select ((menu:Edit → Procedures → New procedure)), or the following icon.



[Creating an event procedure]

Enter a name for the procedure, trigger and function, respectively, into the procedure setting window. Press the [Template] button if you want template code to be created for the function of the event procedure.

- Procedure name:

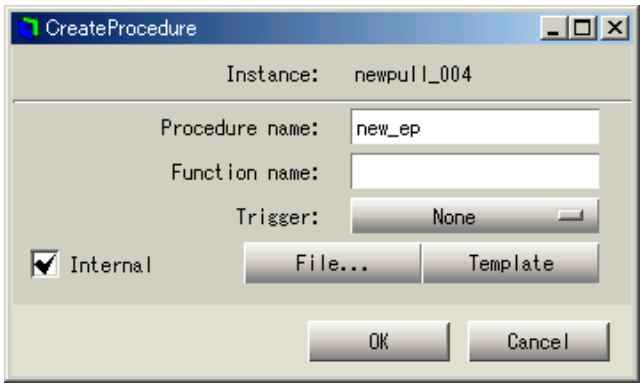
  Enter the procedure name which is used to identify the event procedure.

- Function name:

  Enter the name of a c++ function for the event procedure.

- Trigger:

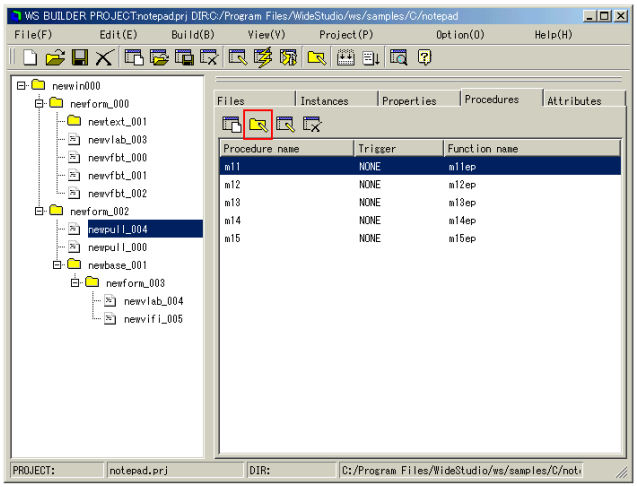
  Choose the trigger for executing.



[The settings of a new event procedure]

5.3.3 How to set up an event procedure

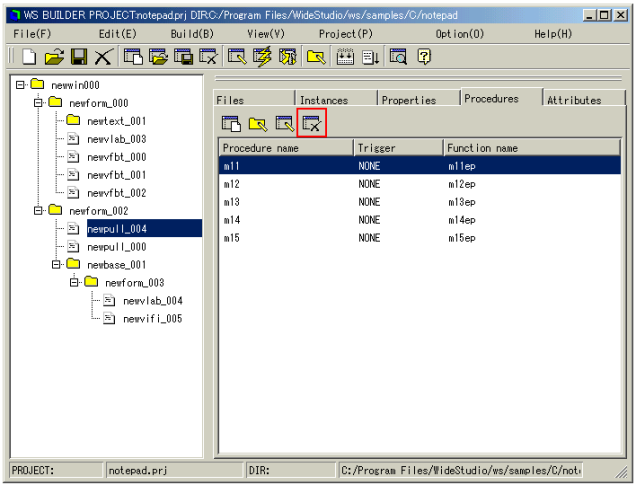
Select the event procedure as per the following picture, then select ((menu:Edit → Procedures → Properties)) or click the following icon to change the properties of the event procedure.



[The settings of the event procedure]

5.3.4 How to delete an event procedure

Select the event procedure to delete, then select ((menu:Edit → Procedures → Delete)) or click the following icon.

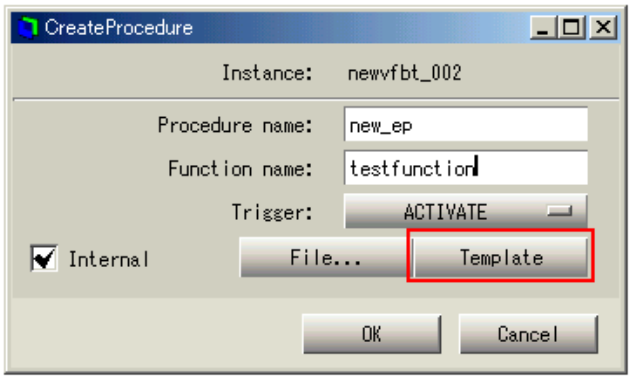


[Deleting the event procedure]

5.4 How to create/edit a function

5.4.1 How to create a template file for a function

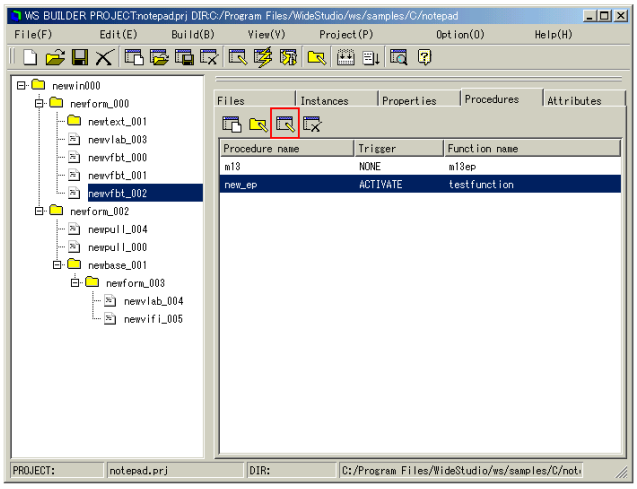
You can create a template file of the function by clicking the [Template] button after filling the function name.



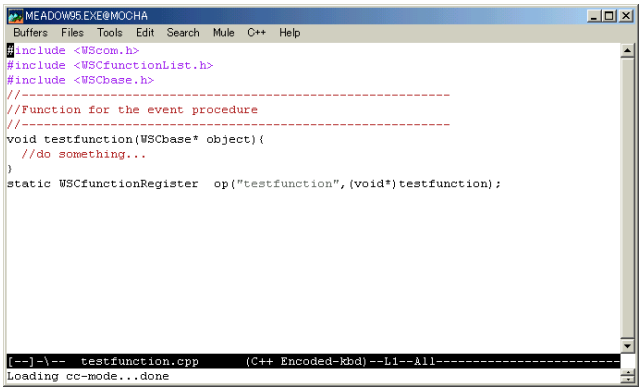
[Generating a template]

5.4.2 How to edit a function

Click the following icon, or double click the procedure name, Then, the text editor is activated, and you can edit the function.



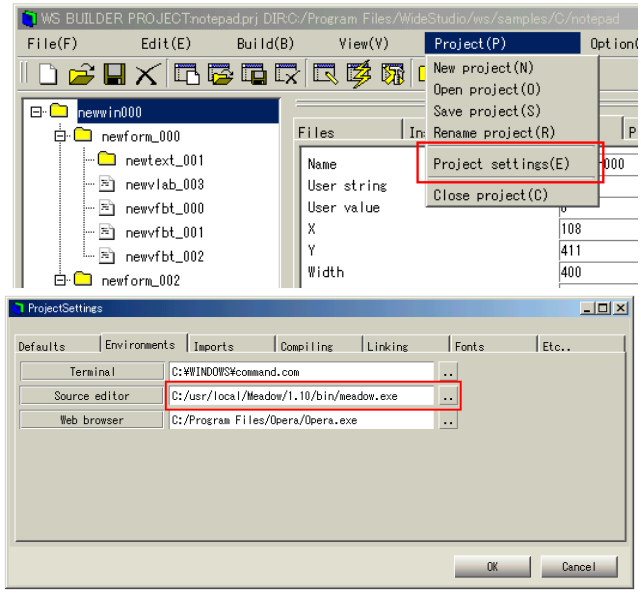
[Editing of a function]



[A view of the editor which stood up]

5.4.3 How to specify your favorite editor

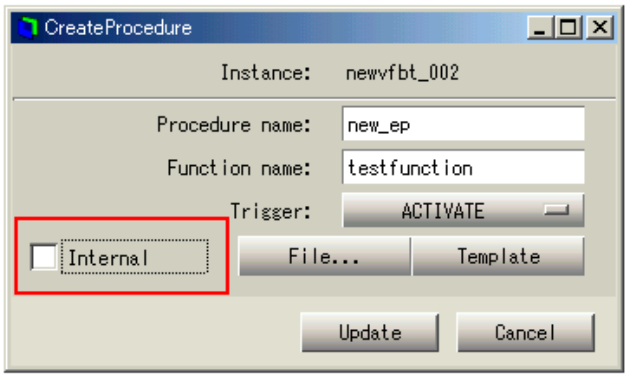
You can use your favorite editor with the environment settings window. Select ((menu:Project → Project settings)), then select the [Environments] tab. Please enter the command line of your editor, arguments included.



[Representation of the environment settings window]

5.4.4 How to use a function which is in a library.

You can use a compiled function which is in a library for the event procedure. Do not select the item from the source but do as follows. In this case, the application builder links the library, but does not compile and link the object file of the function.



[Use a function from a library]

You need to specify a library which includes the object file of the function in the linker options window. See the section:[How to build the project].

Chapter 6

Project

6.1 What is the Project?

A project is the basic unit for the administration of an application which a developer builds with the application builder. A project must be defined for an application in order to build it.

There are many windows and many event procedures in a typical application. The developer needs not manage these source files one by one, the project does it as follows.

- Administration of the program source compilation

  The project compiles the managed application windows and event procedures, and makes the application. There is a function for compiling the updated program sources and for rebuilding the loaded modules.

- Administration of the application windows in the project

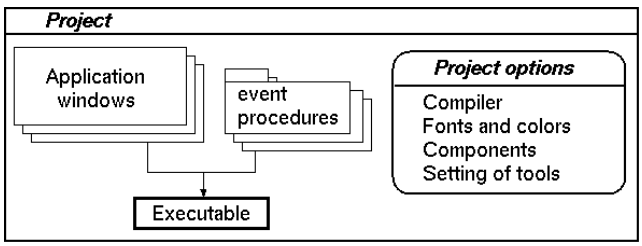
  You can add application windows to the project. The application builder generates corresponding program sources for them, and compiles them automatically.

- Administration of the event procedures for the application windows

  The project manages the event procedures for application windows too. They are compiled automatically to become part of the application.

- Administration of the working environment settings

  The project contains settings such as the default colors or window dimensions, path of the include files/libraries and compiler options.

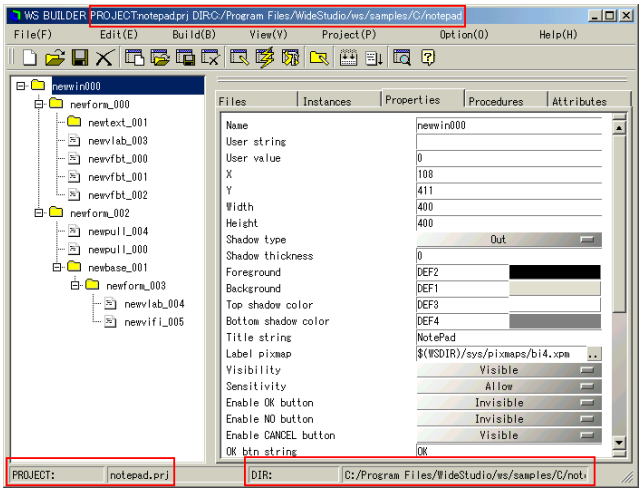


[Project]

6.2 Create a new project, Delete / Save a project

6.2.1 See the project name

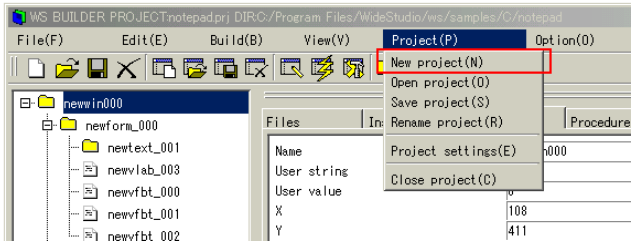
Look at the following figure for where to see the name of the current project. If it is blank, then no project is open.



[Project name]

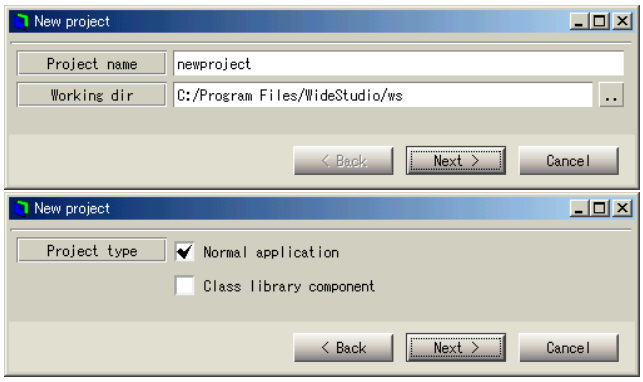
6.2.2 Create a new project

Select ((menu:Project → New project)), or click the following icon to create a new project.



[Creating a new project]

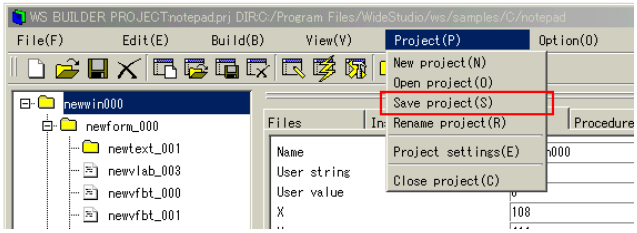
Then, a wizard dialog for the new project appears; input a project name and specify the type.



[Wizard for new project]

6.2.3 Save a project

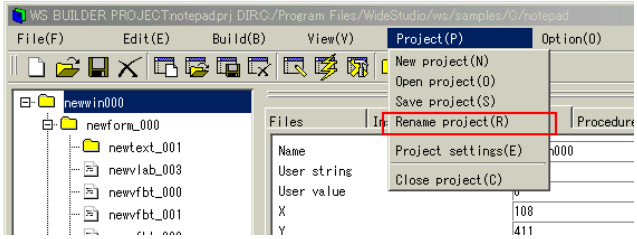
Select ((menu:Project → Save project)) to save.



[Saving a project]

6.2.4 Rename and save a project

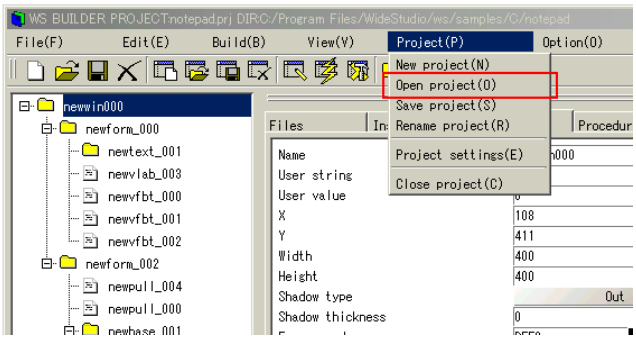
Select ((menu:Project → Rename project)) to rename the project and save it. Then a file selection dialog shows up so you can specify the file name which is used as the new project name.



[Renaming project]

6.2.5 Open an already created project

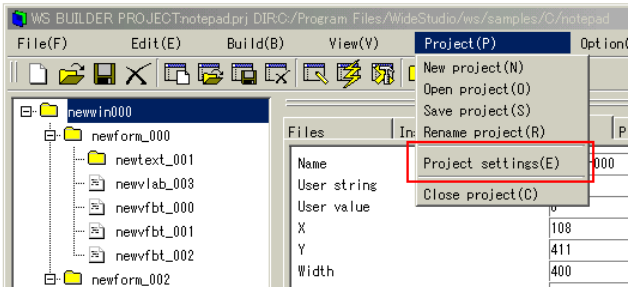
Select ((menu:Project → Open project)) to open an existing, already created project.



[Opening a project]

6.3 Set the project environment

Select ((menu:Project → Project settings)) to set the environment of the project.



[Menu of project settings dialog]

You can set environment variables as follows.

- Default value

  Specifies default values of the WideStudio instance such as window dimensions and so on.

- Tools

  Specifies a terminal, a web browser and a source code editor.

- Import libraries

  Specifies importing of libraries which contain new GUI objects. See section:[How to import libraries which supply new objects].

- Compiler options

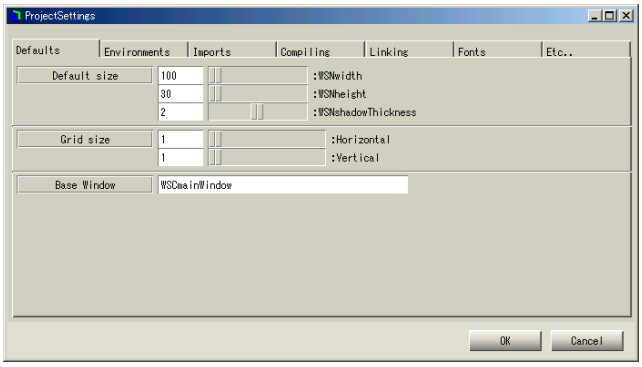
  Specifies options for the compiler, header path, which compiler to use, compilation mode.

- Linker options

  Specifies options for the linker, library path, which library to link, which linker to use, which debugger to use.

- Font settings

  Specifies fonts.



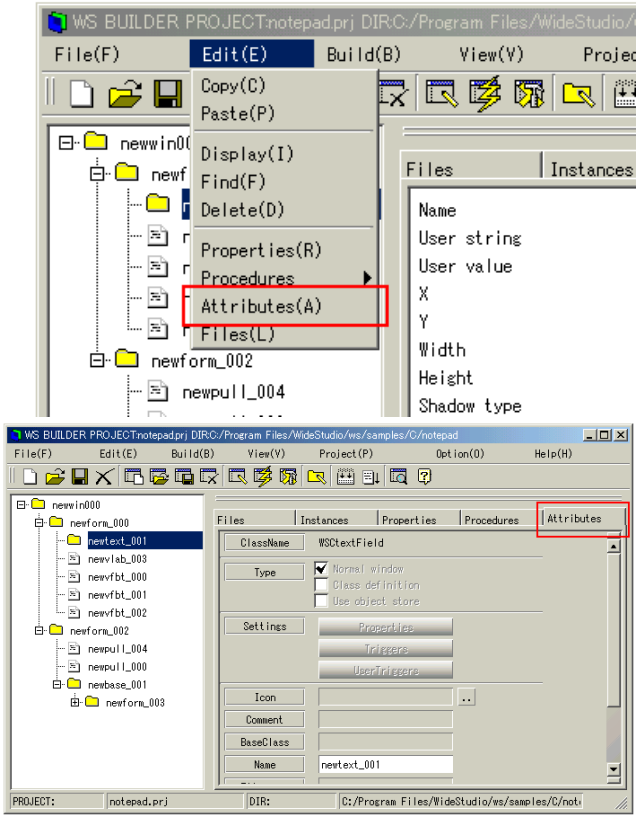
[Project settings dialog]

After setting each item, push the [OK] button to make the change reflect. The settings are stored into a file "[project-name].prj" which is a text file, so you can see it with any text editor / viewer.

6.4 Add an application window to the project

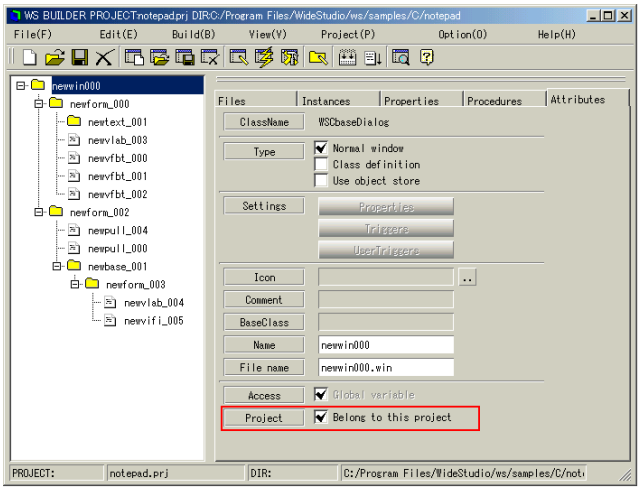
You can add application windows to the project, which will be loaded together with the whole project.

Select an application window to add then select the [Attributes] tab of the inspector.



[Displaying of window attributes]

Check the [Belong to this project] radio button as the following figure.



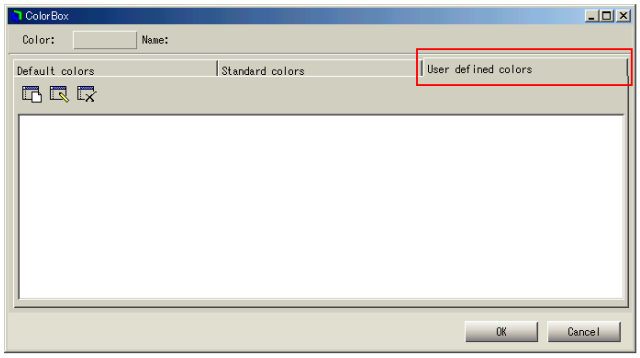
[Add an application window to the project]

Registration information of the application window is stored into the "[project-name].prj" file which is a text file, so you can see it with any text editor / viewer.

6.5 Add a color

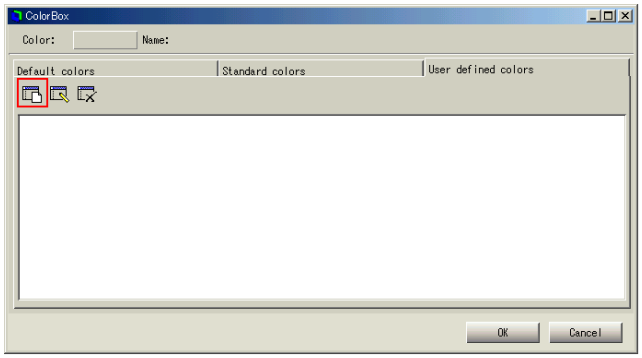
6.5.1 How to add a color

You can add colors to the color selection dialog. Select ((menu:Options → Colors)).



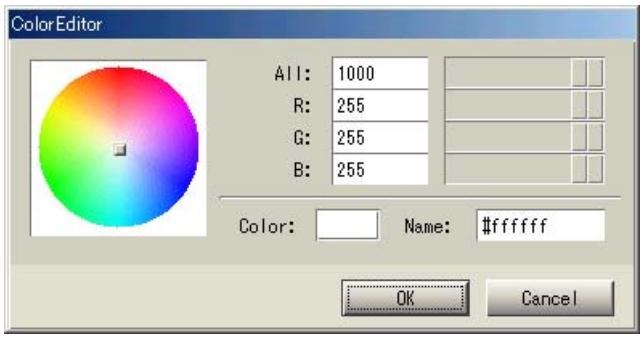
[Displaying of the color selection dialog]

Click the following icon to add a color.



[Displaying of the color editor]

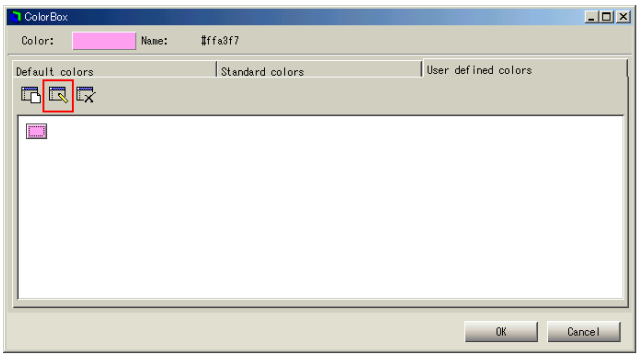
You can set the color by typing the R,G,B value directly or with the sliders or through the color map in the color editor. Once you specify a color, push the [OK] button.



[Color editor]

6.6 How to edit a color

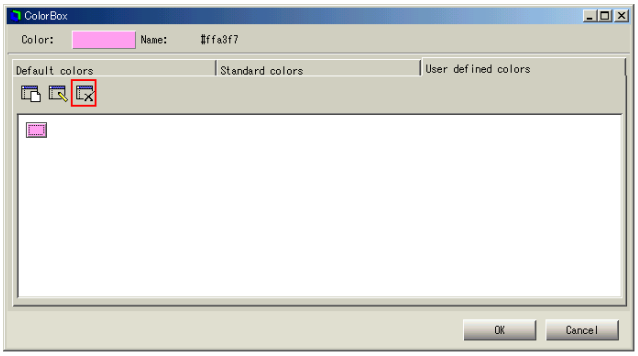
Select the color you want to change in the color selection dialog, then click the following icon; you can only change an user defined color.



[Editing a color]

6.7 How to delete a color

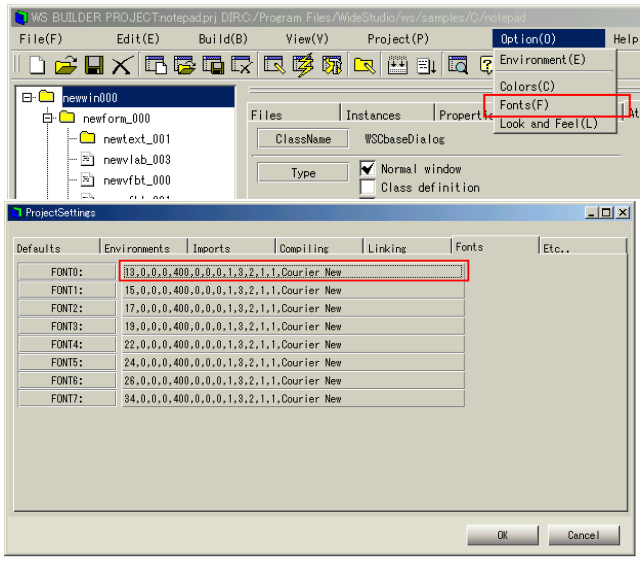
Select the color you want to delete in the color selection dialog, then click the following icon; you can only delete an user defined color.



[Deleting a color]

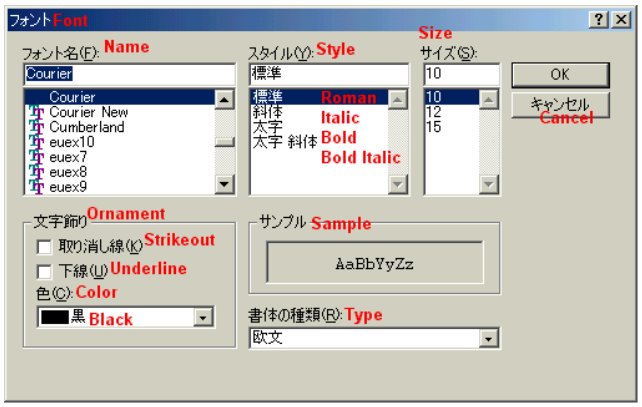
6.8 Set up fonts

You can set up fonts for the project. Select ((menu:Option →Fonts)), then push the button for the font you want to change.



[Displaying the font settings dialog]

Choose your favorite font. The following figure is what you should see on Windows.



[Choosing a font]

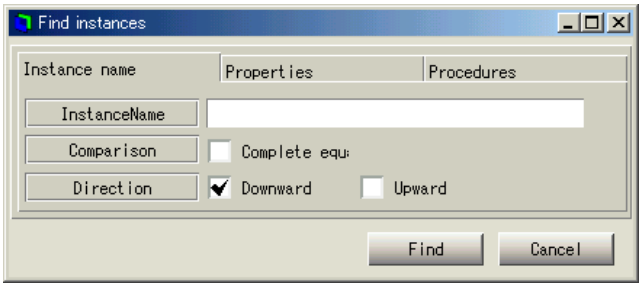
Select the following items:

- 1. font height: dots
- 2. font vendor
- 3. font code
- 4. font family
- 5. font weight
- 6. font slant

6.9 Find application windows and instances

You can find an application window and/or instance in a project by its name, the value of a property, or by event procedure.

Select ((menu:Edit → Find)).



[Search for instances]

There are three ways to search instances:

- by instance name

  You can use the name of an instance as search key.

- by property value

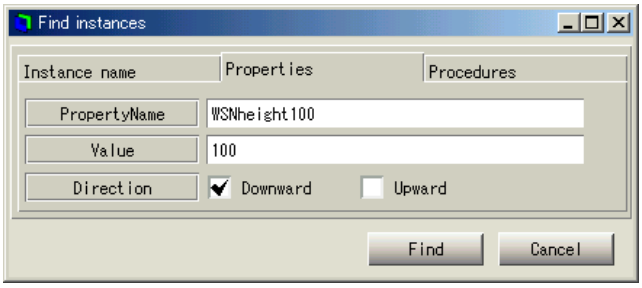
  You can use the value of a property as search key.

  Specify a property in this format: WSNxxxxxx, and a value.

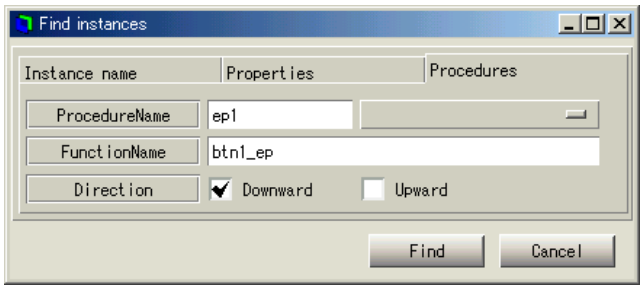
- by procedures

  You can use the name of a procedure, trigger or function name as search key.

Once an instance is found by the application builder, it will be highlighted in the inspector; push the [Find] button for the next one.



[by property]



[by procedure]

6.10 Files in a project

In developing an application with the WideStudio Application Builder, various files will be created. Say there is a project "project1", and it has an application window "newwin000"; the following files are created by the builder.

FILE NAME	DESCRIPTION
project1.prj	Contains the settings of the project.
project1.cpp	The source code of the project.
project1.wns	A list of the application windows.
project1.col	A list of the user defined colors.
newwin000.win	Data for the application window "newwin000"
newwin000.cpp	The source code for the application window "newwin000"

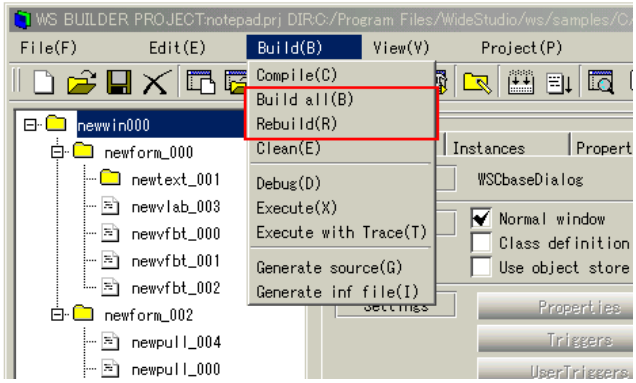
Chapter 7

Compiling / Building

7.1 Build a Project

7.1.1 How to build a project

To execute your application, you need to build a project. The application builder builds compatible project automatically. Select ((menu:Build → Build all)) or ((menu:Build → Rebuild)).



[Building a project]

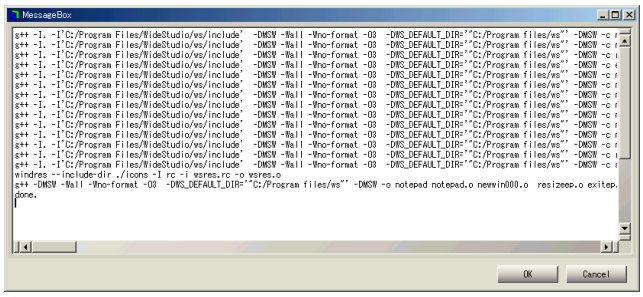
- Build all

  [Build all] command allows the application builder to compile only modified files, making it fast.

- Rebuild

  [Rebuild] command builds the project from scratch compiling all the files. Rebuild command is required when the header file of most of the files is modified.

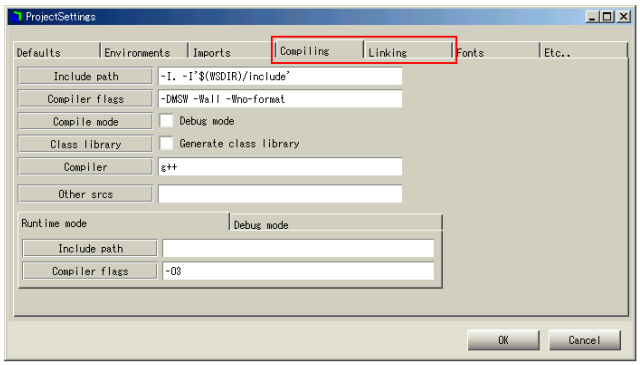
The compilation window looks like the one below when compilation begins. It displays information from the compiler and the linker. It also displays compilation errors if any.



[Compilation window]

7.1.2 How to set header and library path

Select ((menu:Project → Project setting)), then select "compile" or "link" tab.



[Settings for the compiler and linker]

You can set the following options using compile or link command.

- Include path

  Use this option to set the header path.While using a usual compiler of Unix you can set the path as follows:

  -Ipath1 -Ipath2...

- Compiler flags

  "Compiler flags" option sets flags for compiler.

- compile mode

  "Compile mode" option allows you to select normal mode or debugging mode.

- Class library

  You can generate a normal and executable or a Class library (shared library) using "class library" option.

- Compiler

  Compiler option allows you to set your favorite compiler.

- Libraries

  "Libraries" option sets the path of libraries to linking. For a standard linker of UNIX, You can set the path as follows.

  -Lpath1 -Lpath2 ... -llibrary1 -llibrary2...

- Linker flags

  "Linker flag" option sets flags for linker.

- Linker

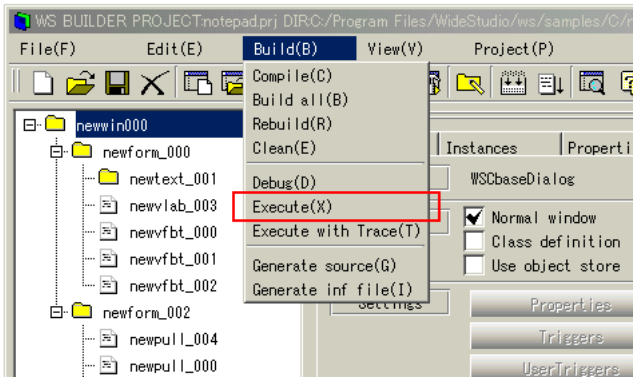
  "Linker" option allows you to set your favorite linker.

- Debugger

  "Debugger" option allows you to set your favorite debugger.

7.2 Execute a compiled load module

You can execute a successful compilation using ((menu:Build → Execute)).



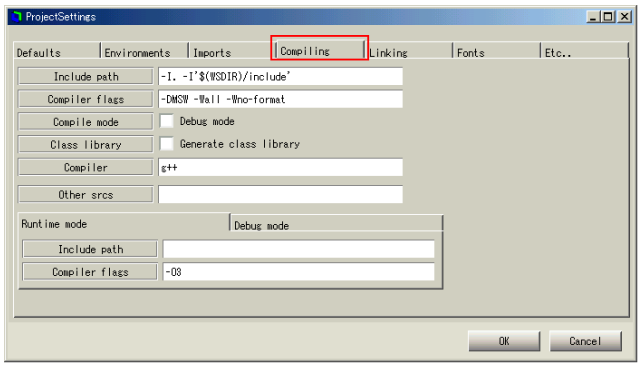
[Executing an application]

To abort the application, select ((menu:Build → Abort)).

7.3 Set compiler options

7.3.1 How to set compiler options

Select ((menu:Project → Project settings)), then select "compiling" option.



[Settings of compiler]

You can set the following options using "compiling" command.

- set path

  Use this option to set the header path. While using a usual compiler of Unix you can set the path as follows:

  -Ipath1 -Ipath2...

- Compiler flags

  "Compiler flags" option sets flags for compiler.

- compile mode

  "Compile mode" option allows you to select normal mode or debugging mode. Debugging mode debugs application with debugger.

- Class library

  You can generate a normal and executable or a Class library (shared library) using "class library" option.

- Compiler

  "Compiler" option allows you to set your favorite compiler.

- Runtime mode

  "Runtime mode" sets options for normal mode only.

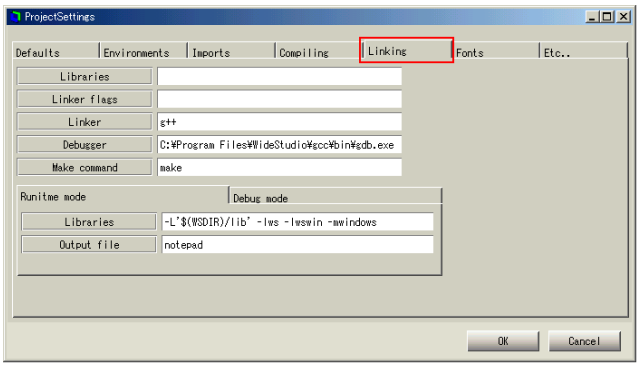
- Debug mode

  "Debug mode" sets options for debugging mode only.

7.4 Set link options

7.4.1 How to set libraries to link

Select ((menu:Project → Project settings)) to set liking options, such as libraries to link.



[Linking options]

You can set the following options using "linking options" window.

- Libraries

  Library option sets the path of libraries for linking. For a standard linker of UNIX, You can set the path as follows:

  -Lpath1 -Lpath2 ... -llibrary1 -llibrary2 ...

- Linker flags

  "Linker flags option sets flags for linker.

- Linker

  Linker option sets your favorite linker.

- Debugger

  Debugger option sets your favorite debugger.

- Runtime mode

  Runtime mode sets options for normal mode only.

- Debug mode

  Debug mode sets options for debugging mode only.

- Output file

  Output file displays the name of the executable file.

7.5 Add Source files

7.5.1 Source file addition setting

You can add any source files to the project by following this procedure. Select ((menu:Project → Project setting)), then select "compile" option. List the object name in the source code that you want to add to the "Additional Object" field.

For example, to add src1.c, src2.c, src3.c to the project, specify as follows:

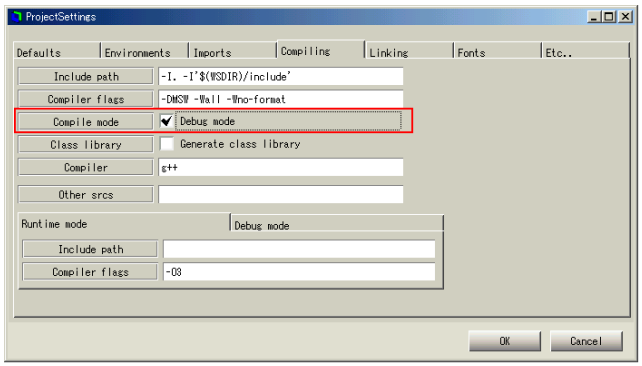
```
src1.o src2.o src3.o
```

7.6 Turn on Debugging Mode

You can build a project with debugging mode. This mode enable us to effectively debug with a debugger.

Select ((menu:Projects → Project setting)), then select "compiling" option, and check the "debug mode".

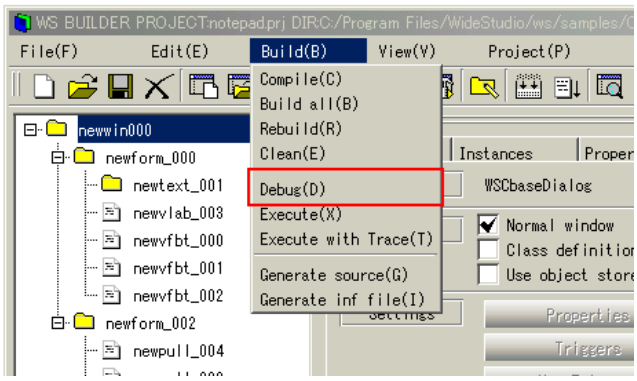
Rebuild the current application, if you change the mode.



[Debugging mode]

7.7 How to Debug an Application

To build an application with debugging mode, select ((menu:Build → Debug)).



[Start the debugger]

Usage of GDB

Enter "run" to execute an application. Enter "where" to stack trace when the program is stopped by error. Enter "list" to see the position of the source.

There are the following commands in GDB.

- Run

  Run command starts the application.

- Cont

  Cont command continues the interrupted application.

- Step/next

  Step / next command runs the interrupted application step by step.

- Where

  Where command Shows stack trace of function call.

- print

  Print command shows the value of variables or functions.

- list

  List command shows the current position of source.

- break

  Break command adds a break point. Enter it as follows.

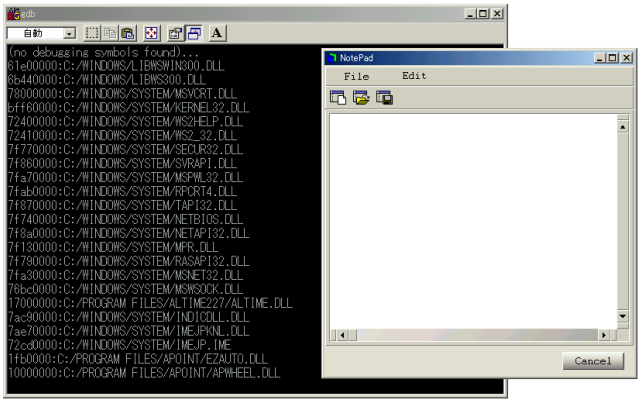
  Break file.cpp:XXX

  Break CLASS: FUNCTION ()

  XXX is a line number.

- cntl-C

  Cntl-C interrupts the application.

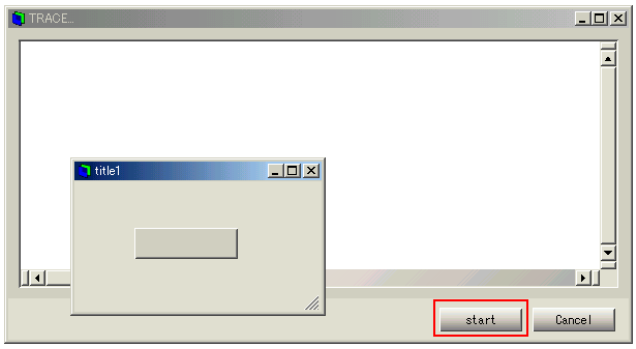


[Debugging the application]

7.8 Trace Debugger

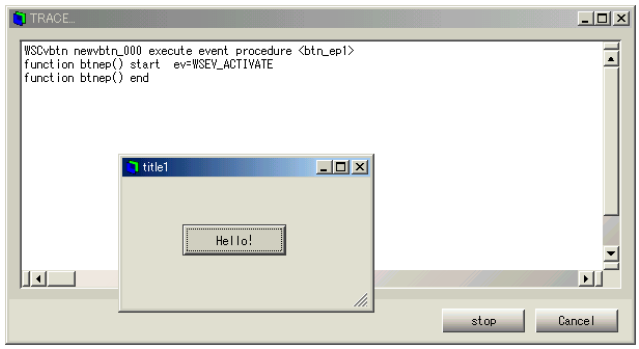
7.8.1 How to use the trace debugger

Use the trace debugger to see the execution status of the event procedures executed. Fist of all, build the application, and select ((menu:Build → Trace Execution)), then the trace dialog appears as follows:



[Trace Dialog]

Push [start] button on the trace dialog to start tracing. When an event procedure is activated, the following outputs are shown in the trace dialog. When you want to stop tracing, push [stop] button.



[Trace output]

You can find which event procedure has a bug when the application terminates abnormally without the following trace message:

```
function functionname( ) end
```

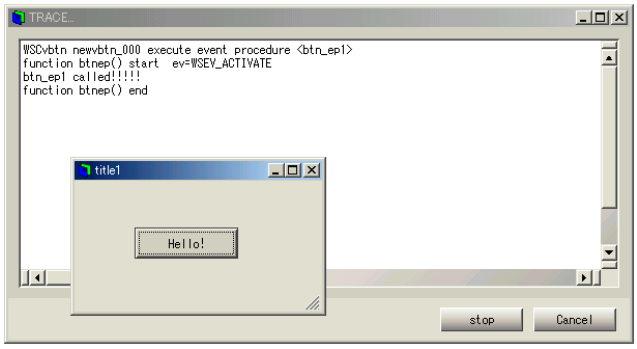
where "functionname" is the name of the function of the event procedure.

Further more, you can show a trace message from your application using WSGFtrace() function as follows (A):

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-----------------------------------------------
//Function for the event procedure
//-----------------------------------------------
void btn_ep1(WSCbase* object){

  object->setProperty(WSNlabelString,"Hello!");
  WSCstring string;
  string = "btn_ep1 called!!!!!\n";
  WSGFtrace(string);                          //(A)
}
static WSCfunctionRegister  op("btn_ep1",(void*)btn_ep1);
```

Note that the output by WSGFtrace() is enabled only when the tracing is turned on.



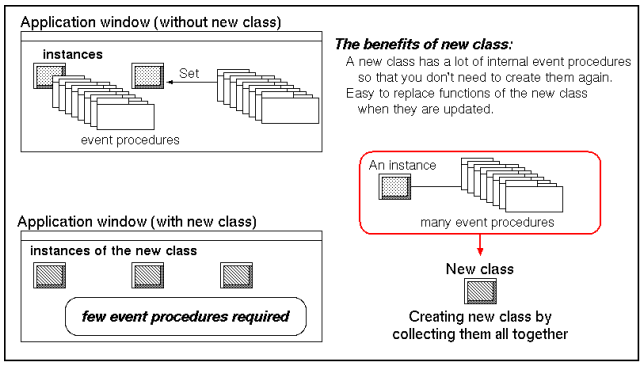
[Trace output by the application]

Chapter 8

Class Application Window

8.1 What is the Class Application Window?

You can make more complicated objects using the C++ object classes of the WideStudio.



You can use a template class generated by the application builder, and it inherits the object class of WideStudio. Developers can define a new class by adding a new member function to it.

The class application window makes create a new object class easy.

- Same development as a normal application window

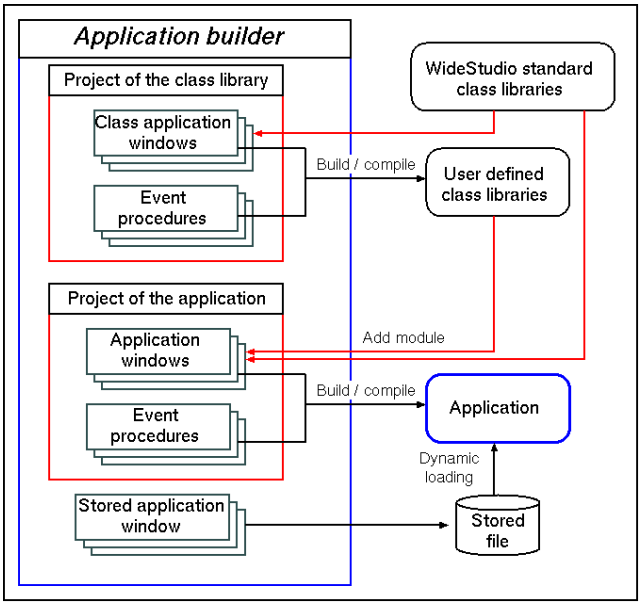
  You can create and edit a class application window using the same methods of a normal one.

- Definitions of new properties

  You can add a new property to the new class by using the dialog, and the application builder generates a source code for it automatically.

- Class library construction

  The application builder can build a shared library for your new classes which can be used by other projects.



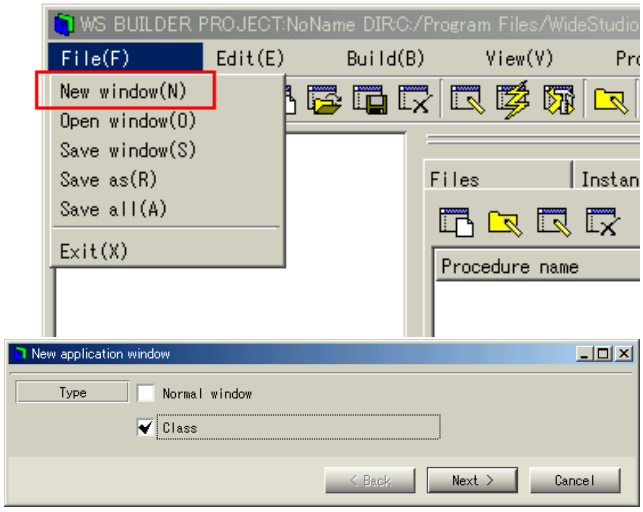
The following table shows the different ways to generate a new class.

Window derivation	The application window becomes the base for a new class.
Composite derivation	Several parts on the application window become the base for a new class.
Component derivation	One object on the application window becomes the base for a new class.

8.2 Create a new class application window

8.2.1 How to create a new class application window

Select ((menu:File → New window)) to bring up the application window wizard dialog, then check the [Class] field.



[A new class application window]

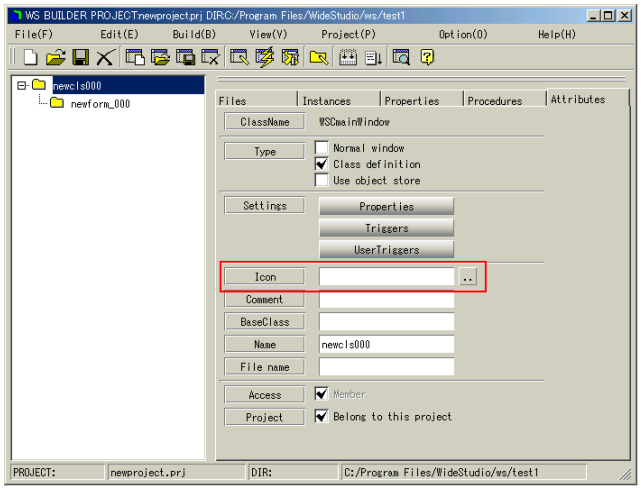
8.2.2 How to select an icon and set the title string of the class

The class application window can be compiled as a class library, which can then be used by other projects. When you load the library project, the classes in the library appear in the [Imported] section of the object box.

You can select an image file for the the icon used by the object box, and can set a title string displayed as the balloon help for the object. If no icon and title string are set, the default icon and title string are used.

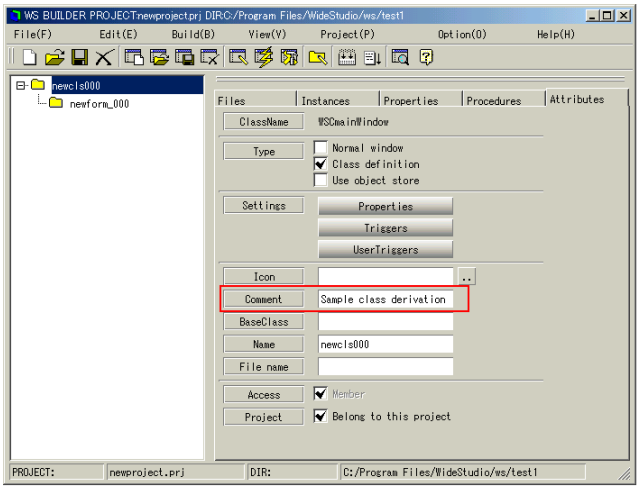
XPM, JPG, PNG and BMP can be used as the format of an icon.

The following figures show the field to select an icon file.



[specify the icon of the class instead of the default icon]

The following figures show the field to set the title string of the balloon help in the Object box.



[Appointment of title string of the balloon help]

8.3 Select the base class for a new class

8.3.1 Default base class

By default, the window derivation is done: the class from the top window of the class application window becomes the base class for the new class. The application builder generates the following c++ source codes, where [classwin] is the name of the class application window.

- "[classwin].h"

  A public header file for the class. You can add new members to this.

- "[classwin].cpp"

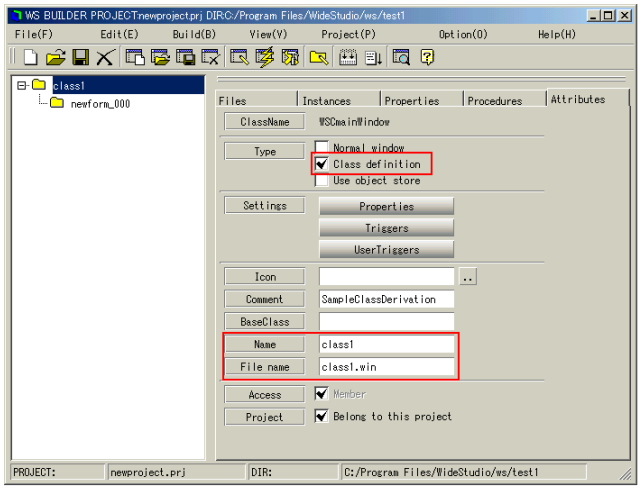
  A public source file for the class. You can add new codes to this.

- "[classwin]P.h"

  A private header file for the class. DO NOT EDIT THIS FILE.

- "[classwin]P.cpp"

  A private source file for the class. DO NOT EDIT THIS FILE.



[Definition of class application window]

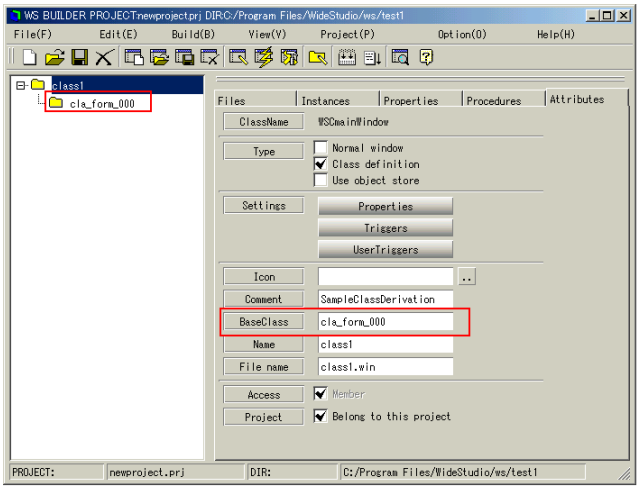
A sample class application window, called "class1", is shown in the figure below and is derived from the class of the WSCwork window, and becomes the C++ class "class1". The top window is used as the base class by default, but you can select a different base class using the [Attributes] section of the inspector.

8.3.2 How to select a base class for a new class

For example of the composite derivation, a part of the class application window, the form "cla_form000", is used as a base class here. By default, the top window is used as the base class, but if another instance of the class application window is selected, it becomes the base class.

(NOTICE) You must re-generate the source code whenever you change the base class. If you have edited the files, save your work into a different file and add it back in after re-generation. To re-generate the sources, delete the following files, where [classwin] is the name of the class application window:

- "[classwin].h"
- "[classwin].cpp"



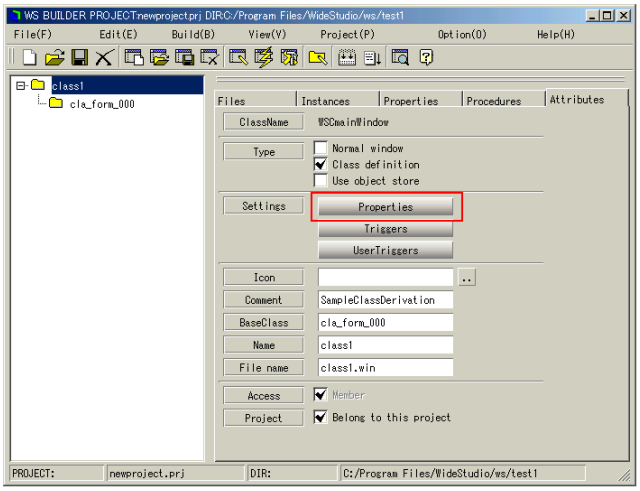
[Selection of a base class]

In this case, a new class "class1" is created from the instance "cla_form000" which is selected as the base class. So, the class "class1" inherits WSCform, along with any child instances on the form "cla_form000".

8.4 Add / Edit / Delete a New Property

8.4.1 How to display the property setup dialog

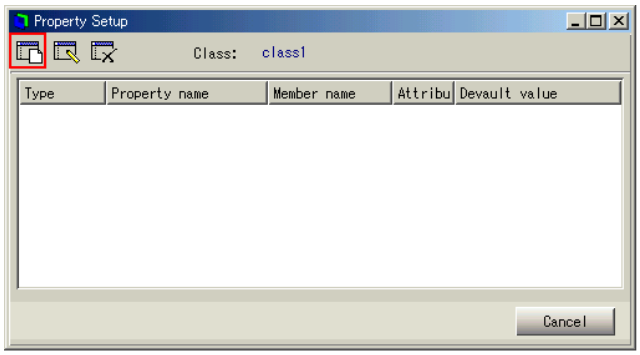
You can add a new property to your class application window from the property setup dialog. Pop-up the dialog by clicking the following button.



[The property setup dialog]

8.4.2 How to add a new property

Click the following icon to display the property creation dialog.



[Creating a new property]

The following items must be set.

- Property name

  Enter a property name beginning with "WSN...". You have to type the prefix "WSN" by yourself.

- Property type

  Select a data type for the variable of the property.

- Variable name

  Enter a variable name for the property. The property uses it to store data.

- Default value

  Enter a default value for the property.

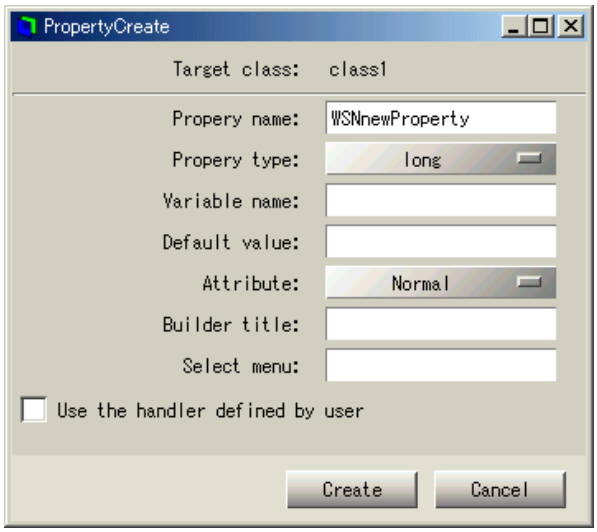
- Attribute

  There are several attributes available.

Kind	Description
Normal	Create a normal property.
Invisible	Create a property invisible to the builder.
Delete	Delete the existing property in the base class.
Change default	Change default value of the existing property in the base class.
Change visibility	Make the existing property in the base class invisible to the builder.

- Builder title

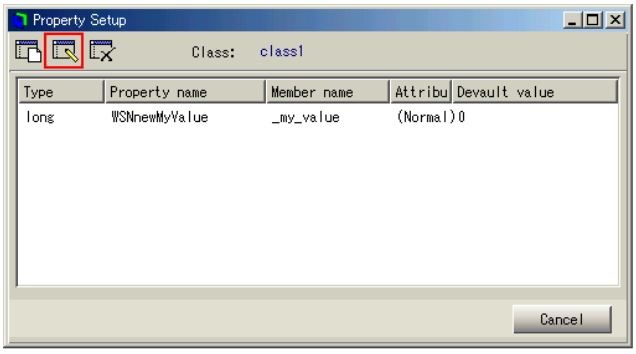
  Enter a property title to be displayed in the inspector, as shown below.



[A view of the property creation dialog]

8.4.3 How to edit a property

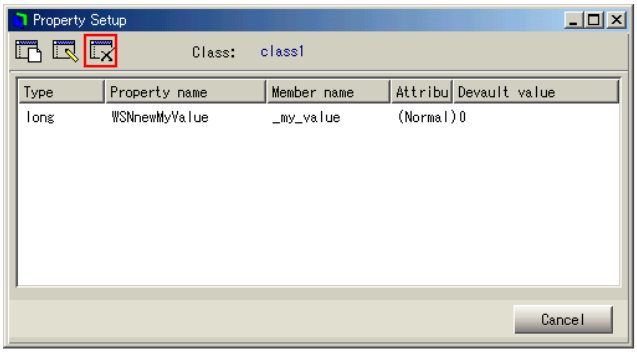
Select a property and click the following icon to edit it.



[Editing of a property]

8.4.4 How to delete a property

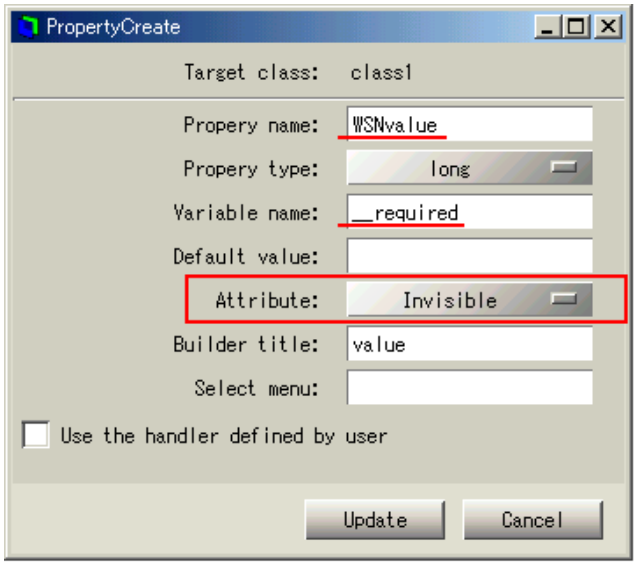
Select a property and click the following icon to delete it.



[Deleting of a property]

8.4.5 How to create a new invisible property

You can create a new property which is invisible to the application builder: the property exists but doesn't appear the property editor on the builder. Select the [Invisible] attribute in the property create / edit dialog.

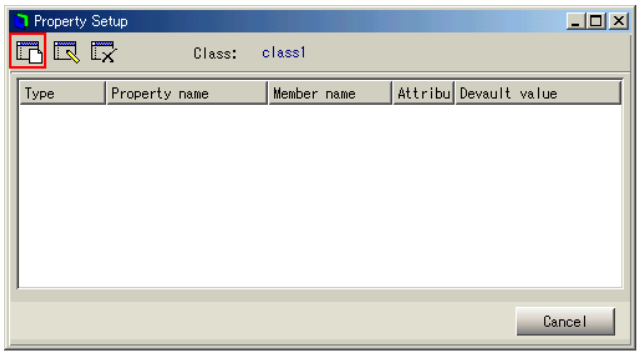


[Create an invisible property]

8.5 Delete / Invisible an Inherited Property

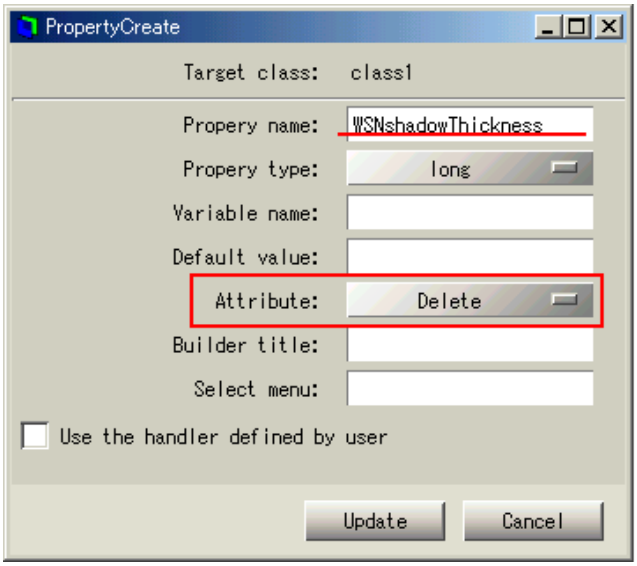
8.5.1 How to delete an inherited property

You can delete an existing property which is inherited from base classes. A new property with the same name and the delete attribute set deletes it. Display the property creation dialog.



[Displaying the property creation dialog]

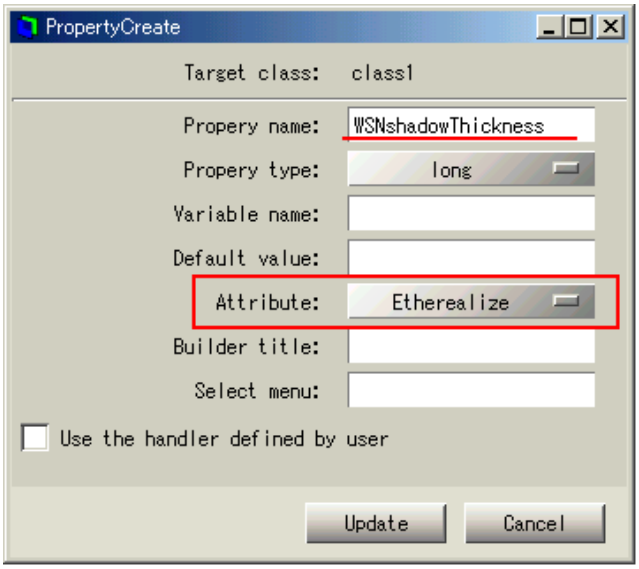
Enter a name of an existing property, then select the delete attribute. The Property type, Variable name, Default value and Builder title are not required. See below, for example, how to delete "WSNshadowThickness" property.



[Deleting an existing property]

8.5.2 How to make an existing property invisible

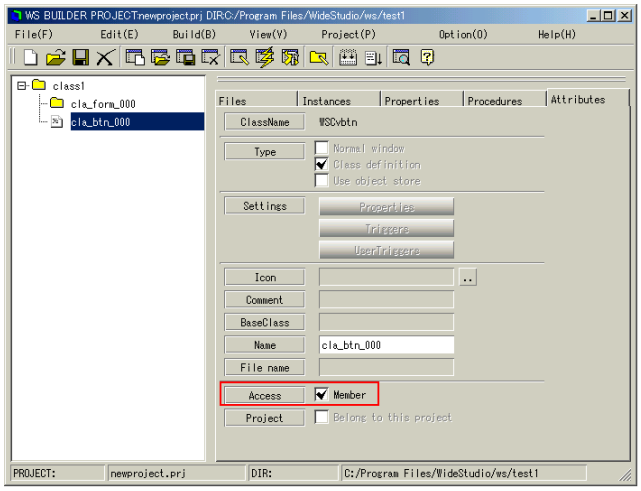
You can make invisible an existing property which is inherited from base classes. With the property name, set the attribute as change visibility. Display the property creation dialog and enter a name of an existing property, then select change visibility attribute. The Property type, Variable name, Default value and Builder title are not required. The following, for example, makes "WSNshadowThickness" property invisible.



[Making an existing property invisible]

8.6 Child instances as members of the class

You can define the child instances as members of the class. Select a child on the inspector, push the [Attributes] tab then check the [Access] radio button.

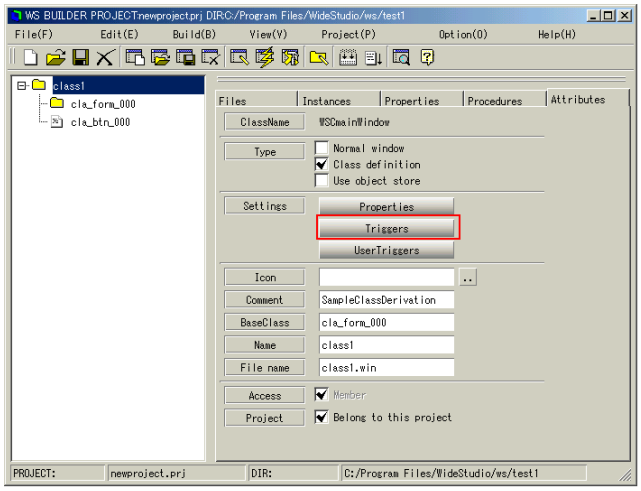


[Definition of members]

8.7 Add / Edit / Delete Triggers

8.7.1 How to display the trigger setup dialog

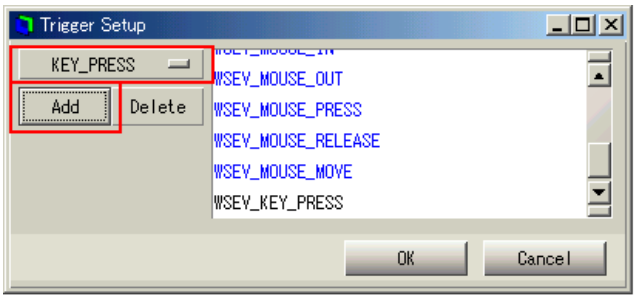
It is possible to add triggers to your class application window with the trigger setup dialog. Click the following icon to display the trigger setup dialog.



[The trigger setup dialog]

8.7.2 How to add triggers

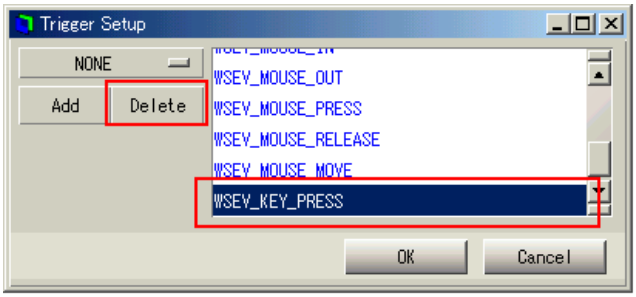
Select a trigger to add from the following menu, click the [Add] button and update the data by pressing the [OK] button.



[Adding triggers]

8.7.3 How to delete added triggers

Select an added trigger from the list, click the [Delete] button then update the data by pressing the [OK] button.

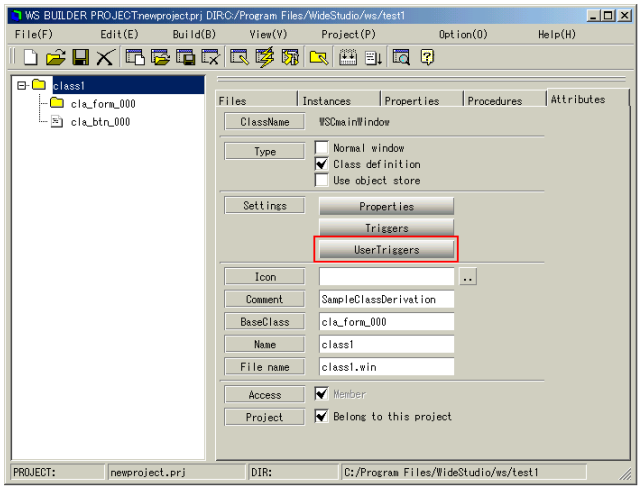


[Deleting added triggers]

8.8 Add / Edit / Delete a User Trigger

8.8.1 How to display the user trigger setup dialog

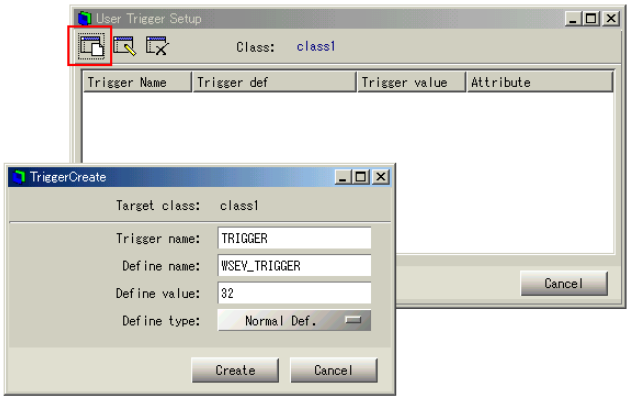
It is possible to add user defined triggers to your class application window with the user trigger setup dialog. Click the following icon to display the user trigger setup dialog.



[The user trigger setup dialog]

8.8.2 How to add user triggers

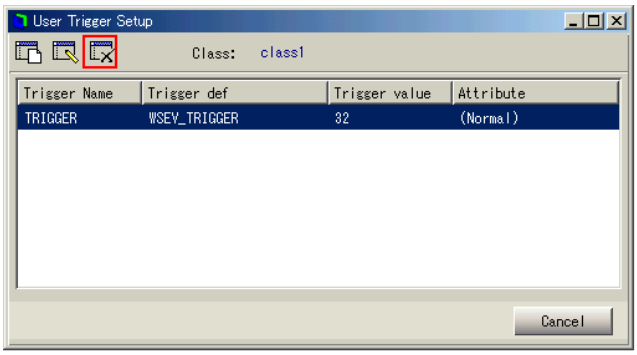
Select the following icon and input the name of the new user trigger, its value, then click the create button.



[Adding user triggers]

8.8.3 How to delete added user triggers

Select an added user trigger in the list and click the following icon to delete it.

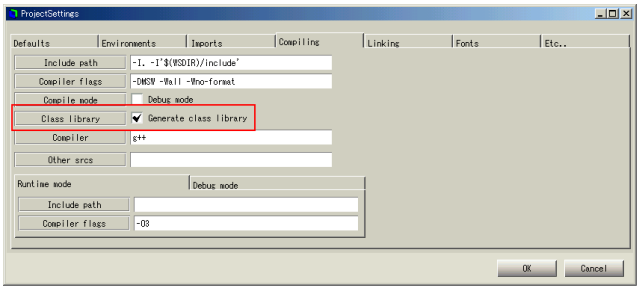


[Deleting added user triggers]

8.9 Create a Class Library

The application builder has a function to create a class library comprising your classes. The output of the project is usually an executable file, but you can select a mode to turn it into a shared library of classes.

Check the [Class library] radio button to make the project output be a class library. The following figure shows the [Compiling] section of the project settings dialog.



[Generating a class library]

In the following, [output] is the output file name which is specified in the [Linking] section.

System	library name
UNIX	lib[output].so (shared library)
	lib[output].a (static library)
Windows	lib[output].dll (dynamic link library)

See section:[How to use the new classes of the import libraries] to use these libraries from other projects.

Chapter 9

Stored Application Window

9.1 What is the Stored Application Window ?

You can store, or export, your application window by the application builder: It then called a "stored application window". A WideStudio application has the function to load the application window dynamically from the stored file, during execution.

This function results in the following.

- Saving of memory, Speeding up start time

  Applications do not contain all of the application windows at the start, thus saving memory and speeding up start time. Display them by loading them from stored files.

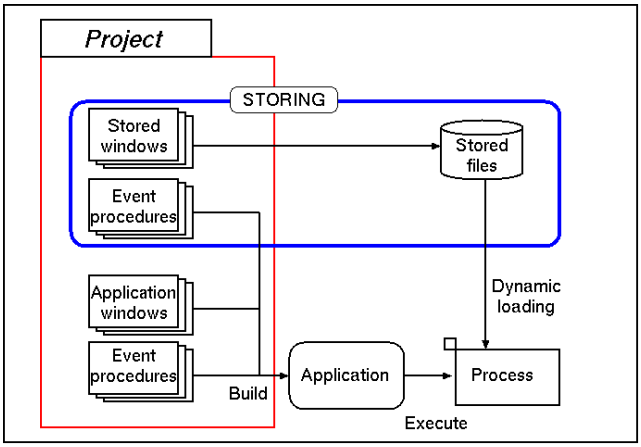
- Switching a part of application window

  An application can load the partially stored windows as data, and display it on its window as a part of the window. This function realizes to display and switch figures like an "Map","Drawing sheet" which is a partially stored window on the window of the application.

- For easy maintenance

  You can share the stored window with other applications, and you can edit it directly, and the window of the application can be updated without compiling. It is a merit for maintenance of an application.

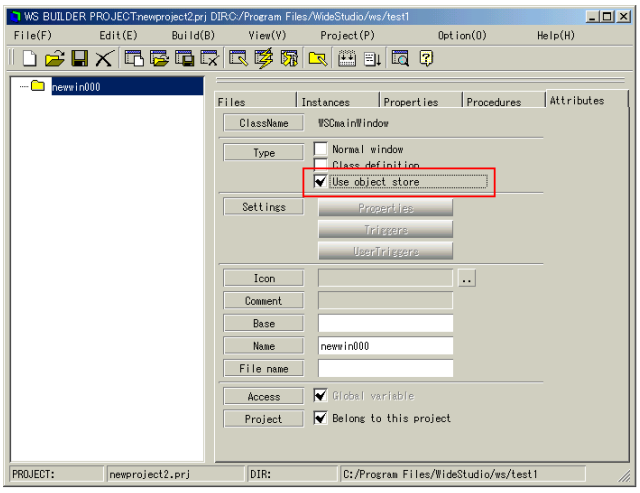
You can store an application window with the application builder. In addition to storing it, you can give the stored window attributes, then the application builder builds an executable which it excludes and stores automatically. The executable is loaded individually with the WSGFloadWindow() function during execution.



9.2 Making an Stored Window

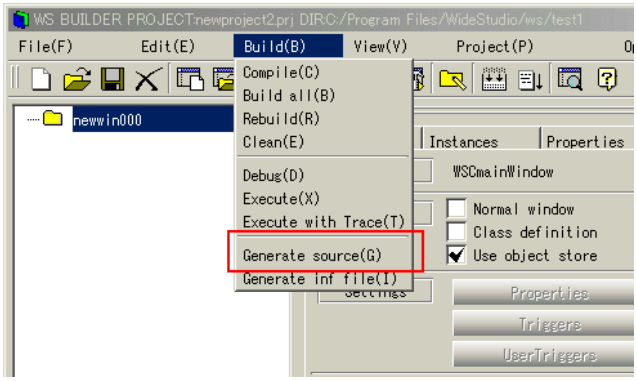
9.2.1 How to save an application window as a stored window

You can save a normal application window as a stored one. Select it with the inspector, select the radio button:[Use object store] of the section:[Attribute].



[Be stored]

Next, select ((menu:Build → Generate source)) as follows.



[Generate a stored application window]

9.2.2 How to add the stored application window to project

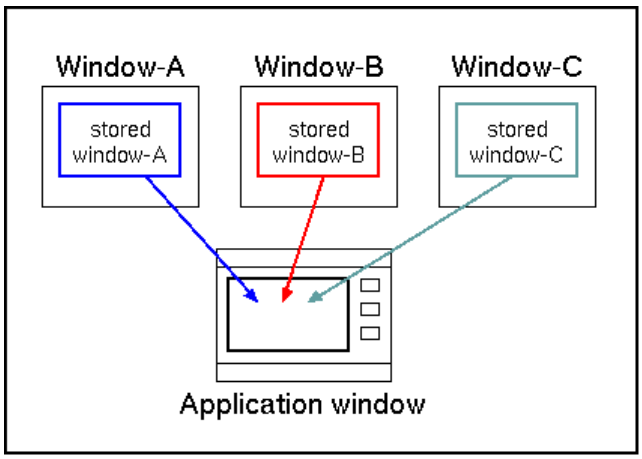
We do not hope to store it manually when building the executable. You can select the stored attribute for each application window of project with the application builder, Select it with the inspector, select the radio button:[Use object store] of the section:[Attribute]. It is the same as the above method to make it be stored attribute. The application builder does not compile it, but builds an executable which it excludes and stores it to an object store file.

The file is "[window name].oof".

The executable then loads individually with the WSGFloadWindow() function during execution. See section:[How to load the stored application window directly from the program] of the Programming Guide.

9.3 Make a Partially Stored Window

You can store a part of application window. The application can load the partial stored window and display it on its own window. A merit of partially stored windows is that application can display them next to next by switching on a same window.

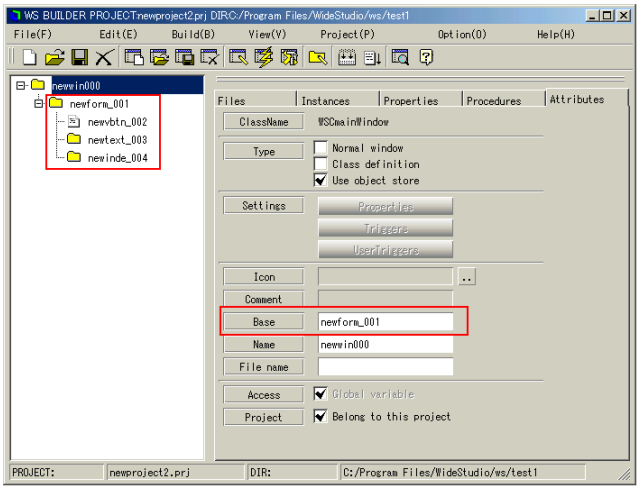


[Displaying partial stored windows]

For example, there are partially stored application window A, B, C which store base is shaded coloring area in the figure. You can load them and display A or B or C on the application window.

Select the [Attributes] tab of the inspector, and write the name of the base instance for partial store in the following field.

If this field is blank, this case is default, the top window is used as the store base instance.



[Field of the store base instance]

See section:[How to load a stored application window directly] of Programming Guide.

Chapter 10

Remote Instance

10.1 What is the Remote Instance ?

WideStudio enables an instance existing on a remote computer running WideStudio to be called on, the same way as if it exists on the local computer. The remote instance feature covers the following:

- Distributed Computing

  An instance (object) of the WideStudio application on a remote computer can be accessed in the same manner as an instance (object) on the local computer, for easy distributed computing. Large scale WideStudio applications can be designed with the functionality to separate tasks into multiple processes, each operating and collaborating with each other.

- Seamless distribution over network

  Instances of the WideStudio on remote computers over a network can be seamlessly referred to without worrying their whereabouts. Agents running on each computer manage remote instances existing on their local systems and exchange information among other agents, keeping any remote instance whereabouts managed.

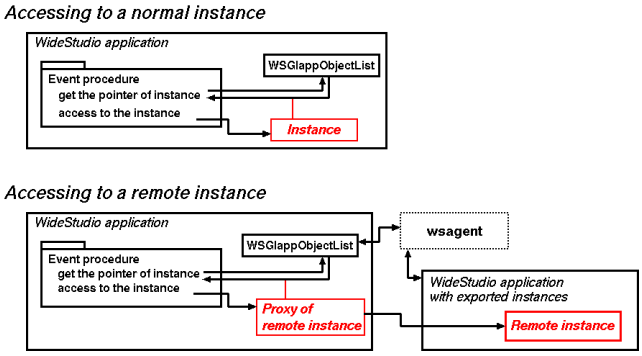
  WideStudio applications automatically access remote instances without allocating to their whereabouts, plugging into the correct remote instances.

- Fault tolerance improvement

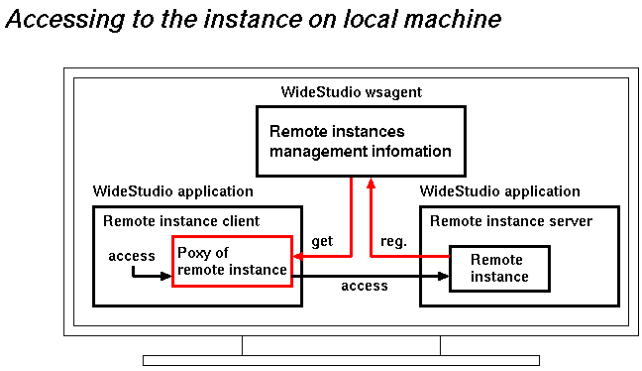
  WideStudio applications can be run with multiplexing. For example, when one application out of the multiplex goes down, an agent detects it and interchanges the remote instance with another WideStudio on a remote computer.

  As for the WideStudio application, it can access the remote instance without being aware of its exchange by multiplexing.

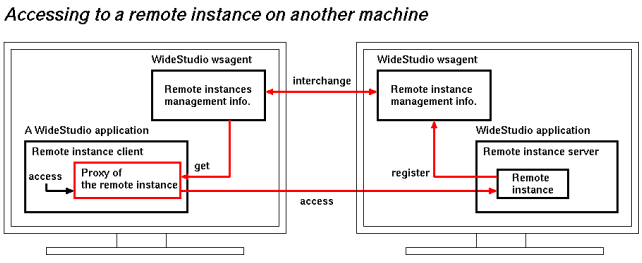
Accessing a remote instance is performed by using a remote instance acquired with an object management, in the same way of accessing an usual instance acquired by WSCGIappObjectList() object management.



In calling a remote instance existing on the same computer, the agent running on the same computer obtains a remote instance to be called.



In calling a remote instance existing on a remote computer, the agents running on each computer exchange information of remote instances to get and be called.



10.2 Start up the WSAgent

The agent plays an important role in realizing the remote instances feature. It provides information on the whereabouts of remote instances for clients who want to access the instance.

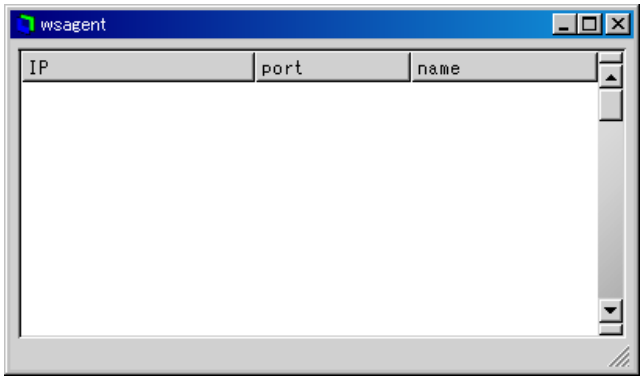
The agent 'wsagent' should be run on the computers before the remote instance feature is used. The command is

```
% wsagent
```

for Unix / Linux,

```
% wsagent.exe
```

for Windows.



[wsagent : the agent for the remote instance]

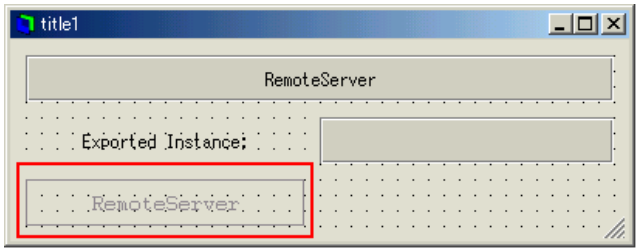
10.3 Construct a Remote Instance Server

A remote instance server is no different from a usual WideStudio application and is to publicize remotely as usual instances existing within applications.

By publicizing as remote, the publicized instances are registered on the agent automatically to be accessed from external applications.

10.3.1 How to make an application a remote instance server

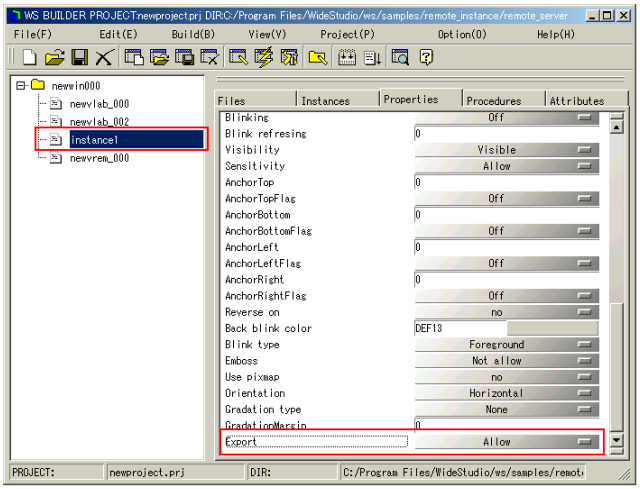
In order to get a WideStudio application to run as a remote instance server, the WSCvremoteServer class is used. The application can become a remote instance server by placing a WSCvremoteServer class instance in the Object Box's NonGUI section on any application window.



[WSCvremoteServer Placing]

10.3.2 How to publicize an instance as a remote instance

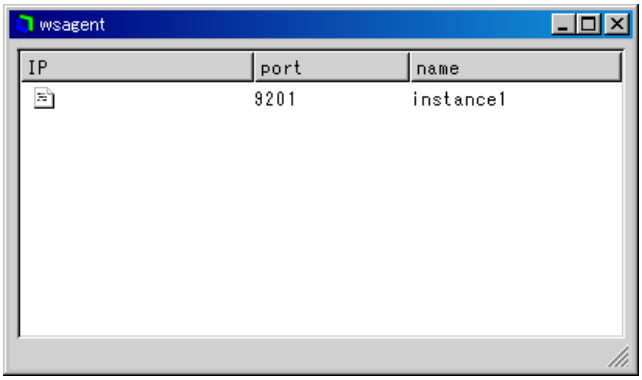
When you want to publicize an instance existing a WideStudio application to external, enable the "Export" property of the instance.



[Publicizing an instance as a remote instance]

10.3.3 How to start up an agent

Starting the agent that manages remote instances publicized on a remote instance server.



[Starting up the agent "wsagent"]

10.4 Summary

10.4.1 Before accessing remote instances

The WideStudio application can access remote instances existing on a remote instance server, the same way as instances existing on that local computer.

First of all, in order to access remote instances, you will need to know the whereabouts of the remote instance on a computer by asking the agent.

An agent manages remote instances existing on a remote instance server.

10.4.2 Before accessing remote instances

Before accessing a remote instances, the WSCvremoteClient class is used. WSCvremoteClient class supports communication to the agent who manages remote instances.

Only actions to communicate with an agent are to be placed on the WSCvremoteClient existing in Object Box's NonGUI section on any given application window of the application.

10.4.3 How to access remote instances

Specifying a name, a remote instance can be accessed via a virtual remote instance given by object management. Please refer to Remote Instance Edition in the Programming Guide for more details.

Acknowledgement

Special thanks to: F. Plesoianu and M. A. Mota Jr..
Thanks to: D. G. Phillips, M. Sudarsan, T. Richards and P. Thorne.

Last modified : November 20, 2002