Introduction to ActiveX™ Component Wizard for Java™

The ActiveX Component Wizard for Java makes it easy to turn a Java class into a COM object. The public methods of the Java class are exposed to COM automation - so you can manipulate your Java class from any COM automation container or scripting language.

Follow the steps the wizard presents, and the intermediate files will automatically be created for you. After the wizard is finished, it will suggest some changes for your original Java source code file. You will need to modify the Java source file, recompile it into a Java class, and copy it to the <*windows*>\JAVA\LIB directory where <*windows*> is your Windows 95 or Windows NT installation directory. This procedure is covered in more detail in **Additional Steps**.

Input Files

To run the wizard, you must specify a Java class file. If you started the wizard by selecting it from the Tools menu in the Visual J++ environment, this file will already be specified. The wizard assumes that the class name will be the same as the .Java source file you are currently editing. For example, if you are editing SIMPLE.JAVA, the wizard will assume you want to convert SIMPLE.CLASS.

Enter the name of the .CLASS file in the edit box. You can use the browse (…) button to select the file. The specified file must be a valid Java class file for the wizard to continue to the next step.

This step allows you to specify the Java class that you wish to convert to an ActiveX component.

Create or specify the IDL file

Yes, please.

Choose this option to allow the wizard to create a new COM interface file (IDL from the specified Java class file).

No. Use the interface definition specified below.

The specified interface definition will be used.

Once a Java class file name is specified, the user can select the "Next" button and successfully create a COM object: the default options will correctly guide you.

By default, the wizard creates an Interface Description Language file (IDL) which describes the public methods of your class. If you have an existing IDL file you created or one created from a previous wizard session, you may wish to skip the conversion and specify the existing IDL file instead. To specify an IDL file, select the "**No, thank you**." radio button. Enter the name of the IDL file in the edit box. You can use the browse button to find the IDL file.

If you ask the wizard to create an IDL file for you, the wizard prompts you for the **CLSID** in the Class Identifier step.

If you ask the wizard to use an existing IDL file, when you click Next, the wizard will compile that IDL file to create a type library (.TLB file). Then, the wizard processes this type library using the Java Type Library Wizard. The wizard prompts you for more information in the **Selecting the Coclass** step.

The Java class-to-IDL conversion will skip Java types which cannot be converted to IDL types (see **Java-to-IDL Conversion Table**). The wizard will only convert methods in the class you specify. It will not expose members of base classes that your class extends.

For information detailing the conversion, see **Java Class Example**.

For Advanced Users:

Use the option to specify your own IDL. This allows you to fine-tune your COM interface. For example, you can modify your IDL to hide public methods that would normally be exposed by removing lines from the IDL file.

Java Class Example

Here is a simple Java class:

```
public class simple {
   public float div (int x, int y) {
      return (float) x / (float) y;
   }
}
```

If the SIMPLE.JAVA file is compiled to a Java class and run through the wizard, the following IDL file (SIMPLE.IDL) will be produced:

```
[
   uuid(7371a241-2e51-11d0-b4c1-444553540000),
   helpstring("simple Type Library"),
   version(1.0)
]
library simpleLib
{
   importlib("stdole32.tlb");
   [
      uuid(7371a240-2e51-11d0-b4c1-444553540000),
      helpstring("Isimple interface")
   ]
   dispinterface Isimple
   {
      properties:
      methods:
      [ helpstring("div method"), id(2) ]
      float div ([in] long p1, [in] long p2);
   };

   [
      uuid(7371a242-2e51-11d0-b4c1-444553540000),
      helpstring("Csimple Object")
   ]
   coclass Csimple
   {
      dispinterface Isimple;
   };
};
```

Class Identifier (CLSID)

In this step, you select how your class will be identified by COM automation.

To invoke your Java class using COM automation, information must be placed in the Registry so that the automation container or scripting language can find the Java class. First, a unique ID identifying your class is required. This is the class identifier (CLSID). The CLSID and other information about the class must then be placed in the Registry.

If your class does not have an ID specified, one will be created for you by default. If you wish to override the creation of a new unique ID, you may specify another ID.

If the wizard detects that your class has a CLSID specified, the wizard will suggest reusing the CLSID.

The wizard will suggest that you modify your Java source file to include the CLSID. This is covered in more detail in **Additional Steps**.

```
private static final String CLSID =
   ("7371a240-2e51-11d0-b4c1-444553540000");
```

If your Java class is modified to specify the ID in this way, it will be selected by default the next time you run the wizard on the class.

Register the Class

Each time the wizard is run, a file will be created which contains information that will register your Java class as an ActiveX component. The file is named <*JavaClass*>.REG, where <*JavaClass*> is the name of the Java class file you are converting.

Choose whether the wizard will enter the necessary information into the Registry. The default is to update the registry with the information in this file.

For Visual Basic Users

When re-running the wizard on the same Java class or when specifying a CLSID already in use, you are not guaranteed that your object will retain project or binary compatibility. If you change your Java class frequently, it is safer to generate a new CLSID each time.

Interfaces

In this step you specify how the IDL for the interface should be created.

Dispinterface or Dual Interface

A *dispinterface* is the simplest form of automation support. Current versions of the Microsoft™ Virtual Machine for Java and Visual Basic require that you use a *dispinterface* to access the new ActiveX component. This is the default choice.

A *dual* interface creates an interface that is both an **IDispatch** interface and a virtual function table (VTBL) interface. A dual interface provides for both the speed of direct VTBL binding and the flexibility of **IDispatch** binding for higher performance.

Note: An incompatibility in the current version prevents the dual interface option from being available for use from Visual Basic or MFC. You can specify that the new VM is to be downloaded to a client machine using the <OBJECT> tag in HTML. For example:

```
<OBJECT
CLSID = "clsid:08B0E5C0-4FCB-11CF-AAA5-00401C608500"
CODEBASE = "http://www.microsoft.com/Java/IE30Java.cab#Version=1,0,0,1">
</OBJECT>
```

Convert IDL to TLB

If you select "**Yes, please**.", after completing the IDL file, the wizard compiles the IDL file to create a type library (.TLB file) to describe the new ActiveX component. This is the default behavior.

The wizard uses the Java Type Library Wizard to create a Java interface to describe the ActiveX component. Your class file must be updated to implement this COM interface. This step is outlined in **Additional Steps**.

If you select "**No, thank you**.", the wizard will simply create the IDL file. Select this option when you wish to modify the IDL file before compiling it. After you modify the IDL file, run the wizard again, specifying the modified IDL file in the **Input Files** step.

Select the Coclass

In this step, you specify the coclass from the IDL file you want to map to the class file you named in the **Input Files** step.

Select the name of the coclass from the drop-down list. The wizard reports an error if there is no coclass in the IDL file. This coclass specifies the CLSID used in registering the ActiveX component. This value is also reflected in **Additional Steps**, in the modification code for your Java source code.

You must move the resulting class file to the Java library directory. This must be done manually. See **Additional Steps** for more information.

When you finish moving the class file to the Java library directory, click Finish to complete the COM Wizard and generate your applet.

Additional Steps

After you have completed all the steps in the wizard, the wizard will process your files. There are a few additional steps that you must do after the wizard completes. This dialog box outlines those steps.

1. Modify the Java source

The Java source file needs to be modified to implement the COM interface. Also, the class file can indicate the CLSID to use if the wizard is run again. The text box contains the modification code for your Java source code. This code is also echoed in the output of the wizard. If you ran the wizard from the Tools menu, the modification code snippet will appear in the output window. You can copy the modification code from either this dialog box or from the output of the wizard. See the example for more information.

Example

For example, in the **Java Class Example** you would replace the line:

```
public class simple {
```

with the following code:

```
import simplelib.*;

public class simple
   extends java.lang.Object
   implements simplelib.Isimple
{
   private static final String =
      CLSID("7371a240-2e51-11d0-b4c1-444553540000");
```

2. Set the output directory

So that the virtual machine (VM) can find the class file, the class file must be placed on the class path. You can do this by copying the resulting file or by setting your output directory in the IDE Build settings dialog.

The recommended location is <*windows*>\JAVA\LIB, where <*windows*> is your Windows 95 or Windows NT installation directory. This directory is the designated location for additional (third-party) class files. Class files placed in this directory will be available to any Java class.

3. Recompile the Java class.

Recompile the Java class to implement the changes described above.

If you did not set the output directory to <*windows*>\JAVA\LIB you must copy the file to this location after recompiling.

Using Your New ActiveX Component

Now that you've created your own COM Object from a Java class, you will want to use it. Based on the **Java Class Example**, here is an explanation of how to use the object from different automation containers and languages.

From Visual Basic

To use your object from Visual Basic

1. Set Reference to include **simpleLib**.
2. In general declarations:

```
Dim x as Csimple
```

3. In Form_Load:

```
Set x = new Csimple
```

4. In some handler:

```
answer = x.div(value1, value2)
```

From VBScript in HTML

1. In the HTML, include an <OBJECT> tag:

```
<OBJECT
ID = SimpleObj
CLSID = "clsid:7371a240-2e51-11d0-b4c1-444553540000">
</OBJECT>
```

2. Within your scripting, use the object:

```
<SCRIPT LANGUAGE=VBSCRIPT>
sub UseSimple
   Answer = SimpleObj.div(value1, value2)
end sub
</SCRIPT>
```

Java-to-IDL Conversion Table

The following Java types will be converted into IDL types:

Java	IDL
java.lang.String	BSTR
com.ms.com.Variant	VARIANT
int	long
short	short
float	float
double	double
boolean	boolean
char	char
byte	unsigned char

Note: The following type mappings are non-symmetric and not obvious. The wizard flags methods that use these Java types as non-convertible.

IDL	Java
VT_DATE	double
VT_CURRENCY	long
[in, out] or [out] parameters	array types

These mappings are based on using MIDL with the **/MKTYPLIB203** switch. This provides the greatest match between IDL and Java types. Both MKTYPLIB and MIDL (without the **/MKTYPLIB203** switch) support some, but not all, of these mappings.

Command-Line Switches

The ActiveX Component Wizard uses the following syntax.

```
JAVAIDL [options] <filename>
```

where the options can selected from the following.

/d Create dual interface descriptions in the IDL file. By default, the wizard will create dispinterfaces (**IDispatch**).

/t[-] Compile the IDL file to create a type library (TLB) file. By default, wizard will compile the IDL file to create a TLB file. Use the minus sign (-) to turn this option off. This is an advanced option. Use **/t-** when you want the wizard to create an IDL file for you which you plan to modify. After you modify the IDL file, use the wizard again, but specify the modified IDL file in the **/IDL** option.

/r[-] Register the class as an ActiveX component. The wizard creates a registry entry (REG) file for the ActiveX component. By default, the wizard will register the component. Use the minus sign (-) to turn this option off.

/IDL *filename* Specify the name of an existing IDL file to use to describe the Java class. By default, the wizard creates an IDL file to describe the Java class as an ActiveX component. Use this option to bypass the creation of a new IDL file.

/TLB *filename* Specify the name of the type library (TLB) file to create from the IDL file. By default, the type library shares the same name as the IDL file, with the .TLB file extension rather than the .IDL file extension.

/REG *filename* Specify the name of the registry entry (REG) file the wizard creates. By default, the wizard names the REG file after the Java class file.

/LibName *name* Specify the library name to use in the IDL file. By default, the wizard names the library after the Java class, with the string “Lib” appended to the name.

/CLSID *guid* Specify the class ID (CLSID) for the new ActiveX component. By default, wizard generates a new CLSID for the ActiveX component, unless one is specified. You can explicitly specify a CLSID using this option. Alternately, if the Java class has a private static final String member named CLSID, the wizard will the value of this member as the CLSID.

/IID *guid* Specify the interface ID (IID) for the new ActiveX component. By default, the wizard generates a new IID for the interface.

/LIBID *guid* Specify the GUID for the type library. By default, the wizard generates a new GUID for the type library.

/CL *filename* Specify the name of the preprocessor to be used by MIDL. By default, it will use CL_VJ.EXE. For more information, please see the note below and **Advanced Users**.

/EXEPATH *dir* Specify the directory to find the executable files (JAVATLB (the Java Type Library Wizard), MIDL and the C preprocessor) needed by the wizard. By default, the wizard expects to find these files on the PATH environment variable. For more information about the executable files the wizard needs, see **Advanced Users**.

/INCL *dir* Specify the directory to find the IDL files needed by MIDL. By default, MIDL expects to find these files in a directory listed in the INCLUDE environment variable. For more information, please see the note below and **Advanced Users**.

/w[-] Run the graphic user interface (GUI) for the wizard. This is the default action of the wizard. Use the minus sign (-) to turn this option off. When the option is turned off, you must specify a class file on the command line.

/v Run the wizard in verbose mode.

Note: MIDL requires the C preprocessor. Visual J++ includes a C preprocessor named CL_VJ.EXE.

(Typically, the C compiler, which includes the preprocessor, is named CL.EXE. The name CL_VJ.EXE avoids naming conflicts with a C or C++ compiler installation on your machine.) By default, the wizard uses the MIDL command-line switch **/cpp_cmd CL_VJ.EXE** to access this version of the preprocessor.

Using an Existing IDL File

When you use the **/IDL** option with **/w-** (do not run the GUI), the wizard processes the IDL file to create Java-callable interfaces. Specifically, it creates Java classes for the interfaces and coclasses in the type library. These classes are created in a Java package in the trusted class path. The wizard needs to know which coclass and ActiveX interface you are modeling. You must also use the **/CLSID** (class ID) and **/IID** (interface ID) options to specify the GUID if they are not unique within the package.

The CLSID you give on the command line will be used in the registry entry (REG) file. If the CLSID given on the command line does not appear in the type library created from the IDL, a warning will be issued.

The ActiveX component must implement a COM interface. The wizard assumes the interface is in the type library generated from the IDL file. The wizard identifies which interface by matching the IID given on the command line to one in the type library. The wizard reports an error if the command-line IID does not appear in the type library.

Incompatible Options

The **/IDL** option is incompatible with the **/LIBID**, **/LibName**, and **/d** options. These options refer to values to use in the IDL file the wizard would generate. The **/IDL** option names a IDL file to use.

The **/IDL** option is also incompatible with the **/t-** option. The IDL file must be compiled to be usable by the wizard.

The **/t-** and **/TLB** options are incompatible since the former tells the wizard not to compile the IDL file, while the latter gives a name for the type library created by compiling the IDL file.

For Advanced Users: Visual C++ and MIDL

The following tools and IDL include files required by the ActiveX Component Wizard have been installed to your <*DevStudio*>\VJ\BIN and <*DevStudio*>\VJ\INCLUDE directories.

The files installed in the INCLUDE directory are:

- `OAIDL.IDL`
- `OBJIDL.IDL`
- `UNKNWN.IDL`
- `WTYPES.IDL`

The files installed in the BIN directory are:

- `MIDL.EXE`
- `CL_VJ.EXE`
- `C1.EXE`

MIDL.EXE is the tool that compiles IDL files to create type libraries (TLB files). It gets standard definitions from the four IDL files in the VJ\INCLUDE directory. Since these IDL files include preprocessor directives (specifically, #include) MIDL requires the C preprocessor to handle these IDL files. Visual J++ includes the C preprocessor in the two files CL_VJ.EXE and C1.EXE. The standard name for the C compiler (and therefore the C preprocessor) is CL.EXE. To avoid naming conflicts the name of the C preprocessor included in Visual J++ was changed to CL_VJ.EXE.

The ActiveX Component Wizard will use these tools and include files since they are specified by **Command-Line Switches** in the tools menu. You can configure the tool to use other copies of these files you have on your hard disk.

If you have both Visual C++ and Visual J++ installed, or if you have an existing MIDL installation, you can override the versions of the tools and IDL files listed earlier in this topic.

To use a different copy of MIDL:

1. Delete MIDL.EXE file from the <*DevStudio*>\VJ\BIN directory on your hard disk. You can reinstall these from the Visual J++ CD later if you wish.
2. Set the system environment variable for the executable path (PATH) to locate the new location of the executable (EXE) files you wish to use. You can use a batch file (AUTOEXEC.BAT or VCVARS32.BAT, for example) to set this value. Make sure the PATH includes not only the directories for these executable files, but other typical directories for commands, such as your windows installation directory. Alternately, use the /EXEPATH command line switch to specify the directory for MIDL.EXE. For more information, see **Command-Line Switches**.

To use a different copy of the C preprocessor:

1. Delete the CL_VJ.EXE and C1.EXE files from the <*DevStudio*>\VJ\BIN directory on your hard disk. You can reinstall these from the Visual J++ CD later if you wish.
2. Set the system environment variable for the executable path (PATH) to locate the new location of the executable (EXE) files you wish to use. You can use a batch file (AUTOEXEC.BAT or VCVARS32.BAT, for example) to set this value. Make sure the PATH includes not only the directories for these executable files, but other typical directories for commands, such as your windows installation directory. Alternately, use the /EXEPATH command line switch to specify the directory for the C preprocessor. For more information, see **Command-Line Switches**.

To use a different copy of the IDL files:

1. Delete the .IDL files listed above from the <*DevStudio*>\VJ\INCLUDE directory on your hard disk. You can reinstall these from the Visual J++ CD later if you wish.

2. Set the system environment variable for the include path (INCLUDE) to locate the new location of the IDL files you wish to use. You can use a batch file (AUTOEXEC.BAT or VCVARS32.BAT, for example) to set this value. Alternately, use the /INCL command line switch to specify the directory for IDL files. For more information, see **Command-Line Switches**.

To use these alternate files in the development environment:

1. Exit Developer Studio.
2. Set the environment variables as instructed above. This includes both creating the batch file and running it.
3. Start Developer Studio again.

   — or —

- Set the switches in the Options Tools to use the new files. For more information on this, see **Command-Line Switches**.

NOTE : If you have set the executable path and include path in Visual J++ by using the Customize Directories option from the Tools menu, these settings will NOT be passed to the ActiveX Component Wizard. You must set the environment variables at the system level before running Visual J++ or specify the directories using command-line switches.

Error Messages

The following error and warning messages are unique to ActiveX Component Wizard for Java (JavaIDL).

Error J0177: **Bad Class File**. The file given for the wizard cannot be opened and parsed as a Java class.

Error J0178: **Bad CLSID Format**. The GUID given for the CLSID, IID, or library ID is not in the appropriate format. The format is 8-4-4-4-12, in hexadecimal digits. For example.

```
12345678-1234-1234-1234-123456789abc
```

Error J0179: **Bad IDL File**. The IDL file given as a parameter cannot be compiled and converted into Java. Both MIDL and the Java Type Library wizard must complete successfully. If either step returns an error, this error is displayed.

Error J0180: **Cannot Use Interface File**. The file specified for conversion describes an interface rather than a class. The wizard requires a class file.

Error J0181: **Cannot Make IDL File**. An error occurred in opening, writing, or closing the interface description language (IDL) file.

Error J0182: **Cannot Make TLB File**. An error occurred in compiling the IDL file to create the type library (TLB) file.

Error J0183: **Cannot Make REG File**. An error occurred in opening, writing, or closing the registry entry (REG) file.

Error J0184: **Cannot Register ActiveX Component**. An error occurred while processing the registry entry (REG) file.

Error J0185: **Too Many Parameters**. The wizard handles one Java class file as input. More than one file was given as command-line parameter.

Error J0186: **Need Parameter**. The wizard must have exactly one class file to run. This error occurs when the wizard is run without UI, but no class file is given.

Error J0187: **Class Java-Callable Wrapper**. The class file specified is a Java-callable wrapper class for an ActiveX component. These files are created by the Java Type Library Wizard.

Error J0188: **Must Specify CLSID**. When running the wizard without the graphic user interface and specifying an input IDL file, you must supply a CLSID for the registry file. For more information, see **Command-Line Switches**.

Error J0190: **IID Not Found in Type Library**. The IID specified on the command-line was not found in the type library created from the IDL file. The ActiveX component cannot be properly created. This error does not appear if the wizard is running the graphic user interface. For more information, see **Command-Line Switches**.

Error J0191: **Cannot Find C-Preprocessor**. The C-preprocessor executable file was not found. MIDL needs the C-preprocessor to compile the wizard-generated IDL file. This is probably an error in the value supplied for the /CL or /EXEPATH options. For more information, see **Command-Line Switches**.

Error J0192: **Cannot Find Include Directory**. The directory specified in the /INCLUDE option cannot be found. MIDL needs the include directory to compile the wizard-generated IDL file. For more information, see **Command-Line Switches**.

Warning J5010: **Ignoring Incompatible Switches**. The set of switches are incompatible. For more information, see **Command-Line Switches**.

Warning J5011: **Method Uses Unsupported Types**. The method includes types that are not supported by the wizard. These types cannot be mapped to IDL types which the Java Type Library Wizard will map back to the given Java type. For more information, see **Java-to-IDL Conversion**

Table.

Warning J5012: **Duplicate Method Name**. The name of the method matches another method in this class. The IDL file will not compile if both methods are included. This method has been excluded (commented out) from the generated IDL file.

Warning J5013: **UUID Not Found in Type Library**. The CLSID or IID specified on the command-line was not found in the type library created from the IDL file. The registry entry (REG) file uses the CLSID given on the command line if provided. The ActiveX component may not be properly registered. This warning will not appear when the wizard is running the graphic user interface. For more information, see **Command-Line Switches**.

Warning J5014: **Different CLSID in Type Library and Class File**. The CLSID found in the .class file does not match the CLSID found in the type library created from the IDL file.