

---

**JADE 5**


---



---

 Jade Development Centre
 

---



---

# ActiveX Automation and Controls

---



# J A D E™

**In this paper . . .**

INTRODUCTION	1
WHAT IS COM / ACTIVEX?	2
COM AND JADE	3
ACTIVEX BROWSER	5
IMPORT WIZARD	6
DATA TYPE CONVERSIONS	11
<i>Variants</i>	12
<i>Safearrays</i>	14
<i>Decimals</i>	15
USING COM OBJECTS FROM JADE	16
<i>Controls</i>	16
<i>Automation Objects</i>	16
<i>Extracting and Loading</i>	17
<i>Registering and Redistribution</i>	17
<i>Automation from JADRAP</i>	18
<i>Automation Event Handling</i>	18
<i>Optional Method Parameters</i>	20
<i>ActiveX Performance</i>	21
EXAMPLE 1 – WEB SITE SEARCHER	24
EXAMPLE 2 – GRAPHING WITH EXCEL	30
EXAMPLE 3 - CATCHING EXCEL EVENTS	35

---

## Introduction

---

The advantage of being able to reuse objects or components in an application has long been recognized. COM provides a standard by which applications can communicate with each other.

In the software universe today, there are many (literally millions) of COM components designed for reuse by other applications. These may be user interface objects (controls) that provide some specialist feature (for example, a speed dial), an interface to a hardware device (for example, a fax machine), or even a whole application (like any of the Microsoft Office applications).

This paper takes a closer look at JADE as a COM client; that is, as an application that can use COM components. We will see how easy it is to make use of thousands of controls and applications that already exist, to provide the functionality required in your application and so reduce the time required to complete your development.

## What is COM / ActiveX?

---

The Component Object Model (COM) is a binary standard designed to support reusable, language-independent and multiplatform components. With COM, it is possible for one component to communicate with another component. These components can be on different threads, processes, or even machines.

ActiveX is Microsoft's implementation of COM. Often, in documentation, COM and ActiveX are used interchangeably. An *ActiveX control* is just a special user interface ActiveX (COM) object designed for use on forms.

COM objects define *interfaces* that expose methods and properties by which clients (user applications) can manipulate the object. In fact, objects can only be manipulated via an interface. Interfaces are immutable, which means that once an interface has been published, it can never change. In other words, methods should never be added, removed, or have their signatures changed. A new version of the object could be provided by a new interface to expose any new functionality while keeping the original interface intact. Objects typically have many interfaces. A method of one interface may access, use, or return another interface.

All COM interfaces inherit from a common IUnknown interface, this provides access to all other interfaces. However the most important interface from JADE's point of view is the IDispatch interface. JADE can only access COM interfaces that are derived from IDispatch. The IDispatch interface provides a way in which applications (in our case JADE) can *bind* (or connect) to the component. COM components that support IDispatch are known as Automation servers.

Every (COM) component has a unique class identifier (CLSID) and every interface has a unique interface identifier (IID). CLSIDs and IIDs are globally unique identifiers (GUID). GUIDs are 128-bit numbers assigned to every component and interface at component development time. COM uses these GUIDs to distinguish between all objects and interfaces. When a component is registered, entries are set up in the System Registry that contain mappings of GUIDs to COM components.

There are two sides to the COM story. A COM server, which provides access to reusable components, and a COM client, which is the user of the reusable component. COM servers come in two flavors, either in-process or out-of-process. An in-process server is a dynamic link library, usually with a **.dll** extension. ActiveX controls tend to use the extension **.ocx**, even though they are just DLLs. In-process servers must reside on the same machine as the client. An out-of-process server implies another application, an executable (**.exe**). Out-of-process servers may reside on another machine within a network, and hence we sometimes see the term *Distributed COM*, or DCOM.

A Type Library is a file that contains descriptions of a components classes, interfaces, data types, and methods. The type library may be a stand-alone file (usually with a **.tlb** extension) or may be embedded within the runtime file (DLL or executable). The contents of a type library may be inspected with a type library browser such as **oleview.exe**, which may be downloaded from the Microsoft web site.

## COM and JADE

To be able to use a COM component from within JADE, a set of classes, properties, and methods are generated that correspond to the objects, interfaces, properties, and methods defined in the type library for that component. In JADE we call this an ActiveX import. Note that we are importing a description and not an object. Once these classes have been created, instances of the COM components can be created and manipulated via standard JADE method calls. The fact that these are COM objects becomes irrelevant and, from a JADE coding point of view, they look like any other JADE object.

The position of a class in the JADE class hierarchy is significant. If a class is a subclass of **Control**, that implies that the class represents a user interface object that can be placed on a form. Therefore, when importing a COM description, the position of the generated class in the JADE hierarchy depends on whether a user interface object (a control) is being imported or not.

Figure 1 shows a JADE class hierarchy with the three ActiveX root classes, **ActiveXAutomation**, **IDispatch**, and **ActiveXControl** before any ActiveX or COM objects have been loaded.

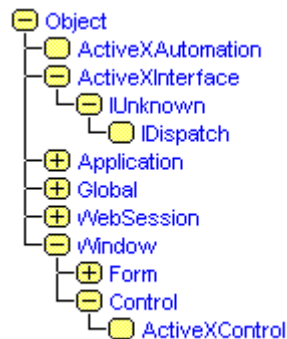


Figure 1 - JADE hierarchy, showing root ActiveX classes

Classes generated that correspond to ActiveX controls are always subclasses of the **ActiveXControl** class, while classes generated for automation servers are always subclasses of the **ActiveXAutomation** class. Any classes generated that correspond to interfaces are subclasses of **IDispatch**.

For example, if we were to import the Microsoft Internet Controls type library (distributed as part of Internet Explorer) we see subclasses of **ActiveXControl** and **IDispatch** generated, as shown in Figure 2.

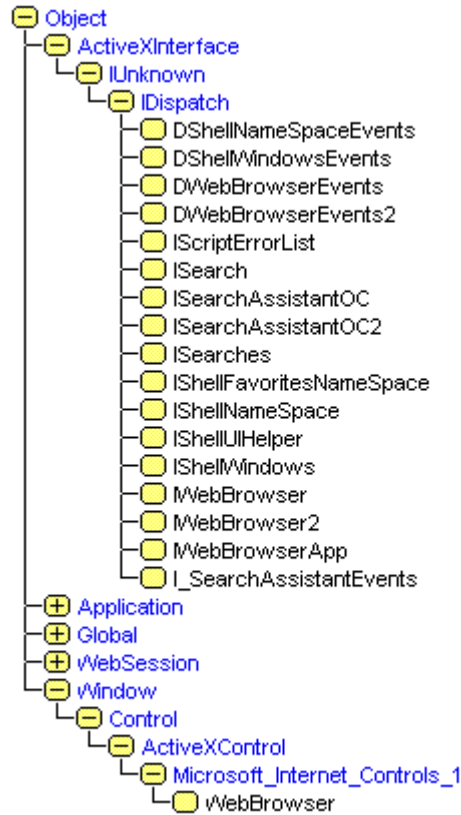


Figure 2 - JADE hierarchy, after importing the WebBrowser control library

You will notice in the above example that two classes were generated as subclasses of **ActiveXControl**, one being a subclass of the other. The first level under **ActiveXControl** is an abstract class that corresponds to the imported type library. Under that are all of the classes (in this case there is only one) that correspond to creatable objects within that library. If we had imported an automation server, similar classes would have been generated as subclasses of the **ActiveXAutomation** class.

## ActiveX Browser

The JADE development environment provides an ActiveX Browser window. It is from this window that you can see what ActiveX libraries have already been imported, and the facilities you can use to import new libraries or extract and remove existing ones. The ActiveX Browser can be accessed from the **ActiveX Libraries** option of the **Browse** menu on the main JADE menu bar. When this window is active, an ActiveX menu appears on the main menu bar. Figure 3 shows a JADE development environment with the ActiveX Browser window as the active window with the **ActiveX** menu displayed.

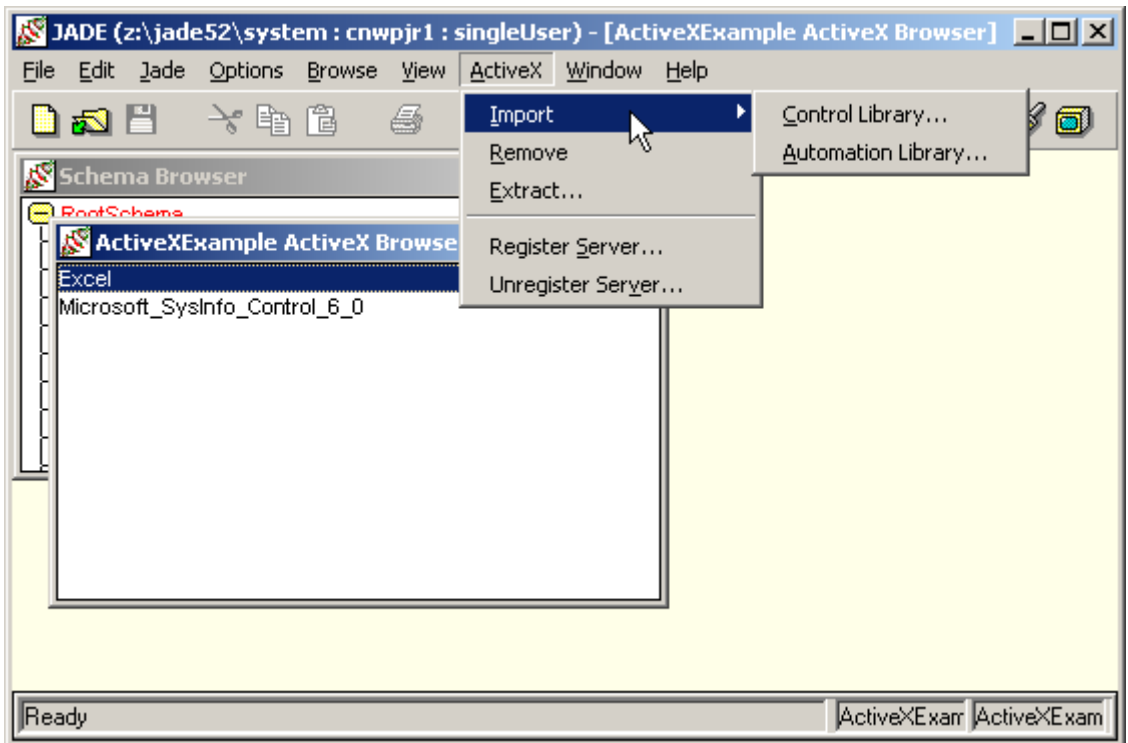


Figure 3 - ActiveX Browser and its menu

## Import Wizard

The process of importing an ActiveX description into JADE is generally a straight forward task. The wizard is accessible from the **Import** option of the ActiveX Browser menu (see Figure 3 above). The **Import** option gives you the choice of importing a control library or an automation library. Both use the same wizard, just that in one case you are loading user interface objects and in the other, automation objects. The first page of the wizard shows all registered type libraries that are available for importing. To appear in this list, a type library must have been registered on your machine. This is normally done as part of the installation of its associated application. Figure 4 shows the first page of the wizard loaded with the automation objects that can be imported.

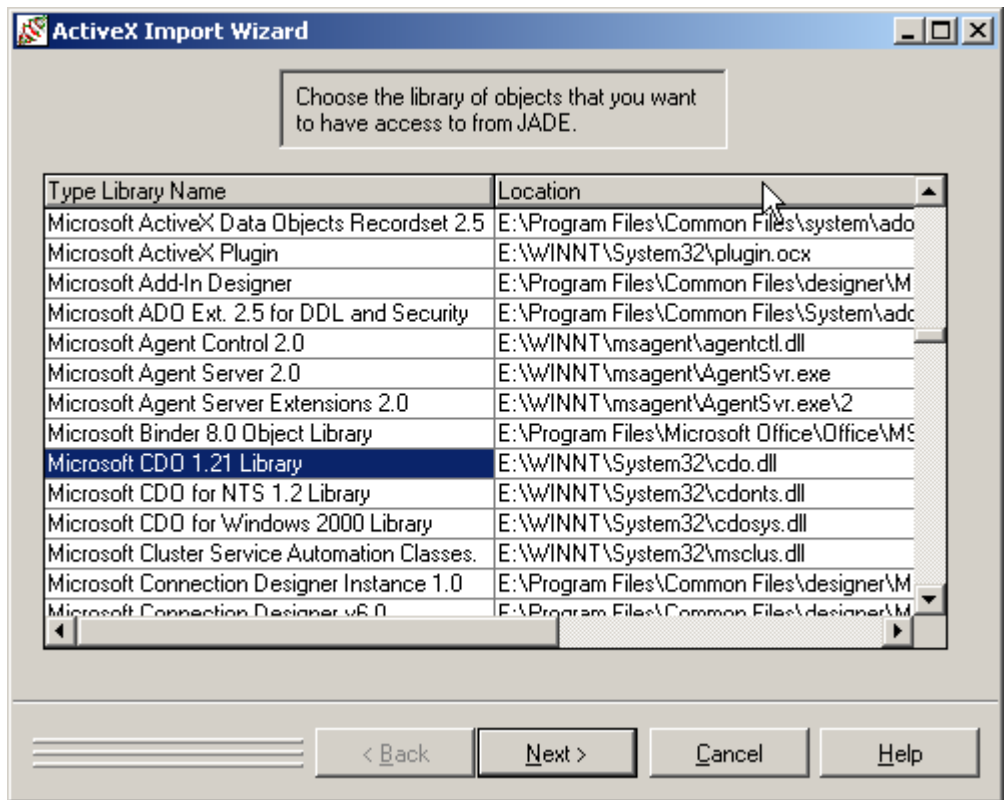


Figure 4 - Import Wizard page 1

If you select a library that has previously been imported into the current schema, you are warned and asked if you want to re-import this library. If so, names you used in the previous import are the default names used for this import.

After selecting the library to load, you are asked for a JADE name for this library. This name is used for the class that is created under **ActiveXControl** or **ActiveXAutomation** (depending on whether you are importing an automation library or a control library). In this example, we are importing an automation object, as shown in Figure 5.

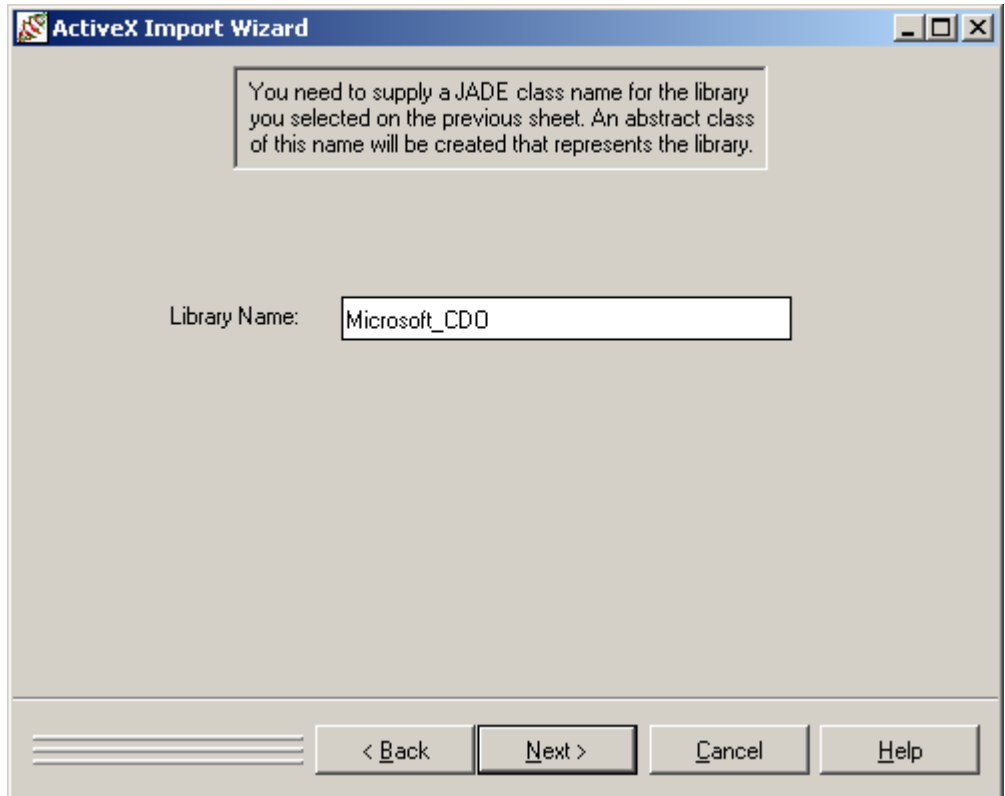


Figure 5 – Import Wizard page 2

On the next page, all of the creatable objects are listed and in our example there is only one. Classes are created for each of these objects. They are subclasses of the library class previously created. You can expand items in the object list to reveal the properties and methods of the default interface.

The type of the properties and type of method return values and parameters depends on the COM type. Unfortunately, as COM was not designed to use the native JADE data types, type conversion is required. Some types are straightforward; for example, a 4-byte COM integer is equivalent to a JADE integer, but there is no simple mapping between a COM currency type and JADE. See the section, “Data Type Conversions”, later in this document, for a discussion on JADE and COM data types.

The methods and properties (of this default interface) are repeated as methods and properties of an interface class, which is created as part of the next page.

Figure 6 shows our one creatable class expanded to reveal its methods. You can use the cells in the right-hand column to provide JADE names for the corresponding objects, methods, and properties. In most cases, you would use the name provided, as this closely matches the COM name and makes it easier to follow the objects documentation. The default names offered may differ slightly from the ActiveX name so they are compatible with the JADE naming conventions.

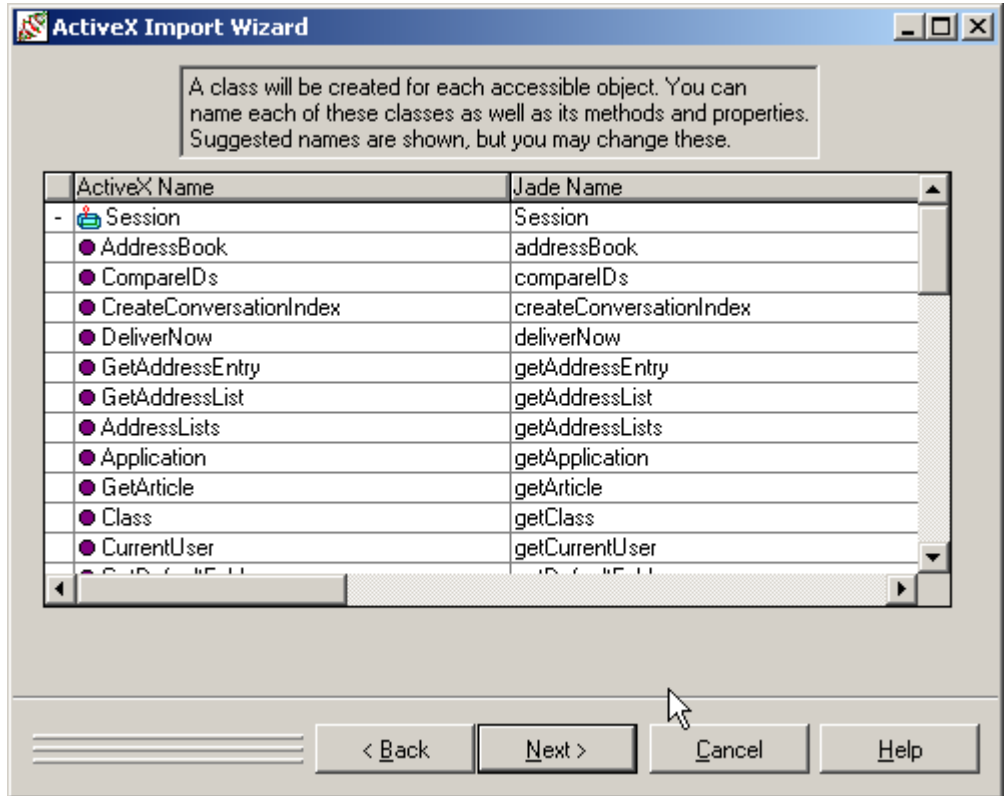


Figure 6 – Import wizard page 3

Any properties that have a COM type of **VARIANT** are replaced by two JADE methods; one to read the property value, the other to update the value. The default name of these methods is built by prefixing the property name with either **get** or **put**. This mapping is done because the COM **VARIANT** type corresponds to the JADE **Any** type but JADE does not allow properties of type **Any**. (See the “Data Type Conversions” in the following section).

In a type library, there may be methods that use duplicated names, that is, two or three methods may have the same name. These are property accessor methods; one to access the property’s value, the others to updated the property. If these accessor methods have parameter types that can be mapped to JADE primitive types (other than **Any**), then these methods are replaced by a property. If these assessor methods can not be replaced by a property, then to satisfy JADE’s naming requirements, two methods are created, one with a name prefixed by **put**, he other prefixed by **get**.

In COM, each object has a *default interface*. The properties and methods of the default interface are duplicated in the JADE class that corresponds to the COM object. This allows the JADE user to access the default interface properties and methods directly from the object class without first having to get the default interface. In the case of controls, the methods of the *default event interface* are also added to the JADE. This makes the use of an ActiveX control identical to using a native JADE control or a user subclassed control.

When importing controls, properties generated for the default interface are marked as design time properties so that they appear in the JADE Painter's Properties dialog (On the "properties of this control" or "specific" sheet ). Most ActiveX controls support *property pages* for setting property values. These can be accessed from the **Property Page** tool button found on the Painter's Properties dialog.

On the next sheet, all interfaces are shown. Again, you can expand the interfaces to reveal all of the properties and methods of that interface, as shown in Figure 7.

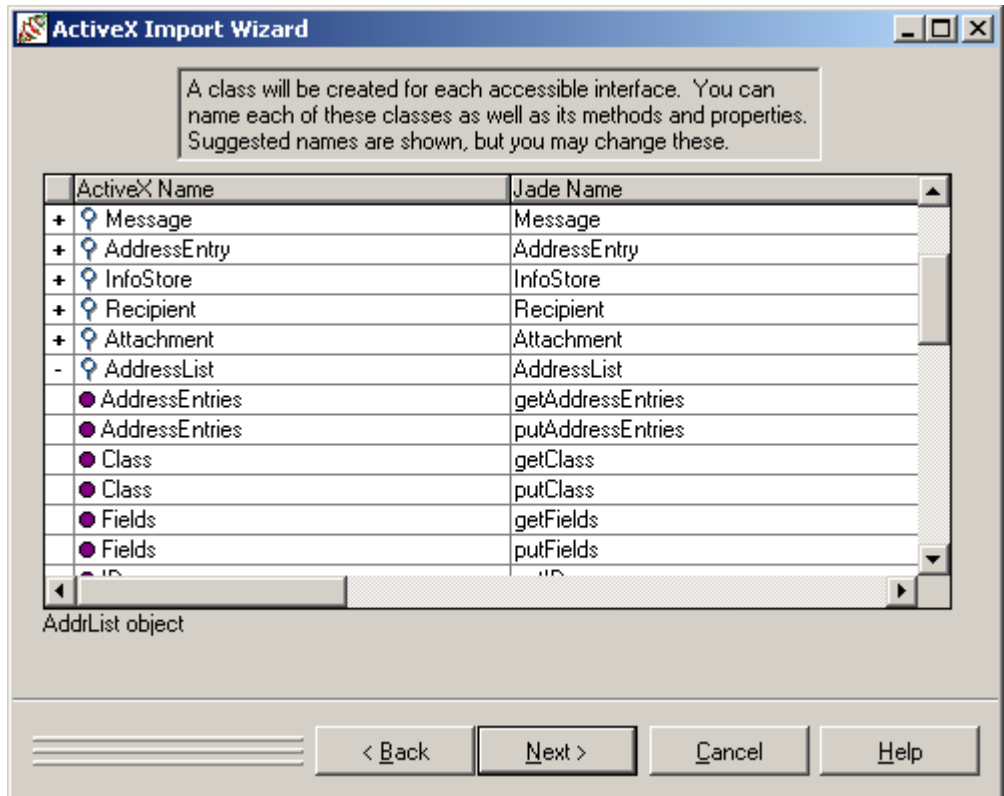


Figure 7 - Import wizard page 4

The last sheet of the wizard allows for the naming of all constants used in the type library. These constants are added to the class that was created corresponding to the library (named on page 2 of the import wizard, as shown in Figure 5)

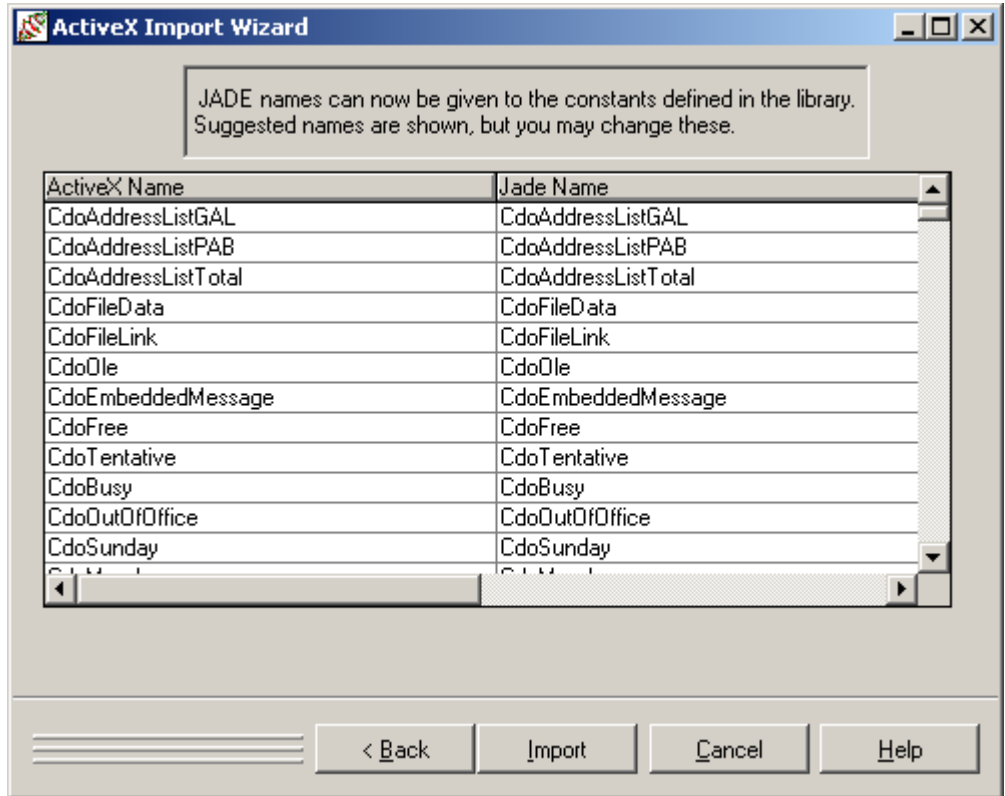


Figure 8 - Import wizard page 5

Clicking the **import** button initiates the import. Classes, methods, properties, and constants named in the previous sheets of the wizard are now generated.

A class is generated for each COM library, object, and interface that is imported. These classes do not have any persistent instances. Each class provides a method of accessing the actual COM library, object, or interface.

On completion of the import, the name of the new library will be shown in the ActiveX Browser window.

## Data Type Conversions

Each programming language has its own set of data types. COM defines a number of types, but for automation, restricts itself to an even smaller set. Most of these COM data types can be safely stored in a JADE type, but when passing data from JADE to a COM object it is possible to exceed the capacity of the COM type. The COM Decimal type is the only case where an automation value could exceed the maximum or minimum values that could be stored in a JADE type.

Automation Type	Description	JADE Type	Conversion Issues
VARIANT_BOOL	Boolean stored as a 16-bit unsigned integer where <b>true</b> = 0xFFFF and <b>false</b> = 0x0000	Boolean	None.
unsigned char	8-bits unsigned	Character	In Unicode, a JADE Character is 16-bits. In this case only the lower order 8-bits are used.
double	64-bit IEEE floating point number	Real	None.
float	32-bit IEEE floating point number	Real	Exception if JADE value causes over/underflow.
long	32-bit signed integer	Integer	None.
short	16-bit signed integer	Integer	Exception if JADE value causes over/under flow.
BSTR	String with a length prefix	String	None.
CURRENCY	64-bit integer	Decimal	Exception if JADE value is not a 64-bit integer
DATE	Fractional number made up of days since Dec 30 <sup>th</sup> 1899. A negative value implies days before Dec 30 <sup>th</sup> 1899	Date	A COM date is from 1 <sup>st</sup> Jan 100 through 31 <sup>st</sup> Dec 9999 while JADE's is from 24 <sup>th</sup> Nov -4713 through 23 <sup>rd</sup> Feb 3268.
VARIANT	16-byte structure that can contain any type	Any	See "Variants" below. JADE properties can't be of type <b>Any</b> so a get/put method pair is created.
IDispatch *		IDispatch	Causes a transient instance of <b>IDispatch</b> to be created. A mapping is maintained between the COM <b>IDispatch</b> pointer and the JADE <b>IDispatch</b> object.

Automation Type	Description	JADE Type	Conversion Issues
Decimal	96-bit unsigned binary integer scaled by a variable power of 10. Gives a number with 28 digits of precision	Decimal	See “Decimals” below. JADE Decimals have 23 digits of precision. An Exception is raised if value exceeds 23 digits.
SAFEARRAY	A variable length structure designed to hold multiple data items of the above types (must all be the same type)	See “Safearrays”	See “Safearrays” below.

Table 1 – Basic COM to JADE data type conversions

## Variants

The COM type **VARIANT** is a 16-byte structure that can contain many different data types. It works in much the same way as a JADE **Any** primitive type. COM method parameters and method return types defined as **VARIANT** have matching method parameters in JADE of type **Any**. But, since JADE does not allow properties of type **Any**, a **VARIANT** COM property is converted into a pair of JADE of methods; one to assess the property value, the other to update it. The names of these methods are based on the property name with a prefix of either put or get.

When passing a value to COM, via an **Any**, the **Any** has a type set internally (based on the data being passed) and it is this type that is used to determine the **VARIANT** type passed onto COM.

Internal Type of Any	Variant Type
Integer	VT_I4
Real	VT_R8
Decimal	VT_DECIMAL
Date	VT_DATE
String	VT_BSTR
Binary	VT_UI1 VT_ARRAY
Character	VT_UI1
Time	VT_DATE
Timestamp	VT_DATE
Any	VT_EMPTY

Table 2 – JADE Any to Variant mapping

Conversely, a variant returned from COM sets the JADE **Any** to a type corresponding to the type of the variant. The table below shows the type mapping used to convert between variants and JADE types.

Variant Type	JADE Type
VT_EMPTY	Sets the value to <b>null</b>
VT_NULL	Sets the value to <b>null</b>
VT_I2, VT_UI2	Integer
VT_I4, VT_UI4	Integer
VT_R4	Real
VT_R8	Real
VT_CY	Integer, Real, Decimal (defaults to Decimal)
VT_DATE	Date, Time, TimeStamp (defaults to TimeStamp)
VT_BSTR	String
VT_DISPATCH	IDispatch reference
VT_ERROR	Integer
VT_BOOL	Boolean
VT_VARIANT	Maps according to this table, based on the variant type
VT_UNKNOWN	IUnknown reference
VT_DECIMAL	Decimal
VT_I1, VT_UI1	Character
VT_I8, VT_UI8	Integer
VT_INT, VT_UINT	Integer
VT_UI1   VT_ARRAY	Binary

Table 3 – Variant to JADE Any mapping

JADE generates an exception if COM returns a variant type that requires JADE to change the internal type of an input-output (io) **Any** method parameter. As an example, imagine a JADE **Any** type containing a string that is passed as an **io** parameter to COM. COM would receive this as a **VT\_BSTR** (see table 2), now if the return value is a **VT\_I4**, JADE would treat this (correctly!) as an Integer type (see table 3). However, JADE can not handle an **Any** type change from string to integer and so an exception is raised.

## Safearrays

Safearrays are variable-length structures designed to hold multiple data items of some specified type. COM limits these types to those listed in table 1 above. Safearrays are mapped into one of the JADE **Array** subclasses, depending on the safearray's type (see table 4).

Safearray Contents	JADE Array Type
VARIANT_BOOL	BooleanArray
unsigned char	Binary
double	RealArray
float	RealArray
long	IntegerArray
short	IntegerArray
BSTR	StringArray
CURRENCY	DecimalArray
DATE	TimeStampArray
VARIANT	<see below>
IDispatch *	IDispatchArray
Decimal	DecimalArray

Table 4 – Safearray to JADE array conversions

Note that a safearray of unsigned chars becomes a **Binary** in JADE and not an array of characters or a string.

Safearrays of variants present a problem. Although the member type of the array (**VARIANT**) is consistent, the content of the variants may vary; for example the first cell may contain a string, the second an integer, and the third a date, and so on. This means that the corresponding JADE array would have to vary the type of each cell to match that of the safearray. JADE does not support arrays of mixed type.

However, JADE arrays can be passed as a safearray of variants (it just happens that each cell is the same data type). These restrictions give rise to the following rules:

1. Method input type parameters defined in COM with a type of SAFEARRAY(VARIANT), that is a safearray of variants, are fully supported.
2. Method out or in-out type parameters or a method return of type SAFEARRAY(VARIANT), cause an exception at runtime if the array contains mixed data types.

## Decimals

JADE **Decimal** type properties when declared (either as a property or a method variable) must specify a precision and scale. For example, to create a decimal method variable of precision 10 and scaled by 2, you need:

```
vars
    decimalExample : Decimal[10,2];
```

However, a COM decimal does not carry around its size information, all we know is that it is a 96-bit number scaled by some power of ten (which gives 28 digits of precision). Therefore a JADE decimal created to correspond to a COM decimal is declared as **Decimal[23]** (the largest possible decimal). You may have to modify this to suit the decimal values actually used by the COM object; for example, **Decimal[23,2]**.

Decimals are the only case where a JADE property may not be big enough to hold all the values used by a COM object. An exception is generated if a number from COM cannot fit into the JADE variable.

# Using COM Objects from JADE

---

## Controls

For a JADE user, using controls could not come any easier. Just go into the JADE Painter and paint your controls as you would for any other control. From the JADE developer's point of view, basic JADE controls, user subclassed controls and ActiveX controls are all used in the same way.

Controls, being user-interface objects, must be run on the presentation client. This means that all of the supported DLLs (usually **.ocx** files) need to be on the presentation client. In addition, the control must have been registered on the presentation client machine (see "Registering and Redistribution" below).

When doing a **create** of an ActiveX control class, only the JADE instance is created, the actual COM object is not created at this stage. The creation of the COM object is delayed until the JADE control is added to a form. This usually happens in one of three ways: when a control is dropped onto a form in the Painter, when a form that contains an ActiveX control is loaded at run time, and when logic calls the **Form::addControl** method to add a dynamically created control instance. Doing a **create** of an ActiveX control class does not, in itself, create a COM instance.

There is a fourth way in which the COM object for a control can be created. For a JADE instance of an ActiveX control, the method **ActiveXControl::createAutomationObject** forces the COM object to be created. A control created in such a way can not then be added to a form, and is used in the same way as automation objects (see "Automation Objects" below). Not all controls support this method of creation. Controls that do are typically ones that become invisible at run time.

## Automation Objects

A **create** of an **ActiveXAutomation** subclass class only creates you an instance of the JADE class (in the same way a create of an ActiveX control only creates a JADE instance of a control). There are two ways in which to create an automation object. First, by using the **ActiveXAutomation::createAutomationObject** method and second, by accessing any property or method on the default interface of the automation object.

Before creating the actual COM object, there are two properties on the **ActiveXAutomation** class that can be used to specify where the object is to be run.

### **ActiveXAutomation::usePresentationClient**

This property has a default value of **true**. This means the COM server associated with the object runs on the same machine as the presentation client. Setting this property to **false**, forces the COM server to run on the same machine as the JADE application server (**jadapp.exe**). Of course, this means you must be running in thin client mode or this property has no effect.

### **ActiveXAutomation::remoteServer**

This property allows you to set the machine name on which the COM server runs. This is a string that contains the machine name either as a UNC name (for example, \\servername) or as a DNS name (for example, 123.4.56.78 or jade.com). This machine does *not* have to be a JADE machine. To use a remote server you need DCOM (Distributed COM) installed and configured by using **dcomcnfg.exe**. On Microsoft Windows NT and Microsoft Windows 2000, **dcomcnfg** is installed by default when you install the operating system. However you need administrator rights to run it. For Microsoft Windows 98 and Microsoft Windows 95, you need to install **dcomcnfg** first.

You can use the above properties together. The **usePresentationClient** specifies which machine the remote requests are made from, and **remoteServer** the machine on which the server executes those requests.

## **Extracting and Loading**

An ActiveX definition can be extracted and loaded into another schema without having to re-import the ActiveX type library. However, to be useful the whole definition of the ActiveX library must be extracted and loaded. Individual ActiveX libraries can be extracted via the **Extract** option of the ActiveX menu. For details, see Figure 3.

An ActiveX library extract creates **.scm** and **.ddb** files. When loading, both the **.scm** and **.ddb** files need to be loaded. Some users wrongly assume that the **.ddb** file is only relevant to form and control extracts but this is *not* the case.

The extract and load only deals with the JADE definition of the ActiveX library. Your extract or load does not move the COM server to or register it on any new target machine.

## **Registering and Redistribution**

All ActiveX objects and interfaces need to be registered in the system registry. This provides a mapping between the CLSIDs (see the “What is COM “ section) and the COM server (DLL or executable) that handles those objects.

Registration is normally performed as part of the installation process of the server. However, you can use **regsvr32.exe** utility program to register or unregister an in-process server. For example:

```
regsvr32 myControl.dll
regsvr32 -u myControl.dll
```

Out-of-process servers (executables) normally provide a command line option (or switch) to perform registration. Generally, this switch is **/regserver** but may be different for some servers. An example is:

```
excel /regserver
```

Registration can also be performed from the ActiveX menu. (For details see Figure 3.) The register and unregister options do not affect in any way the definitions that have been loaded in to JADE. They merely provide a shortcut to system utilities for registration.

If you distribute an application that uses ActiveX controls, you must make sure that the controls are installed on each computer that runs your application. Distribution rights of these controls are determined by the control's creator. Be sure you have the necessary rights before distributing any control as part of a JADE application.

## Automation from JADRAP

Automation methods cannot be tagged with **serverExecution**. However, they can be used in a non-GUI application which is started by an **app.startApplication** method that is part of a **serverExecution** method. In this case, the database server (**jadrap.exe**) must reside on a Windows machine.

## Automation Event Handling

Automation normally consists of issuing commands (property access and method calls) to an automation server. Automation events are a way the server can tell the client what it has done or is about to do. This is the same as a JADE control, say a **Button**, tells JADE that it has been clicked (via the **click** event).

Open the JADE Class Browser of a form class that has controls. In the properties pane of the browser you will find reference properties for the controls. Clicking on any of these control references causes the methods list pane to be filled with all of the events that the control supports. You can then add logic to any of the events. This logic is executed when the event is triggered inside the control.

The JADE implementation of automation events is similar to the way in which control events are handled. Just as we have a reference to a control, we need a reference to an automation event interface. Controls are added via the JADE Painter, but automation event references are added via the **Add Event Reference** option of the Properties menu in the JADE Class Browser. The list of possible event references is a subset of the **IDispatch** subclasses. Event **IDispatch** classes have been marked as event interfaces during the type library import for the server.

Once added, the event reference can be selected. This (like selecting a control reference) displays a list of implementable events in the methods pane of the Class Browser. These events are shown dimmed to indicate that have not yet been implemented. Events can then be implemented by clicking on the greyed event name and adding logic to it.

By default, automation events are not passed on to JADE. Event notification has to be requested. To indicate our desire to receive, and possibly later to stop receiving events, we use the following methods.

- `beginNotifyAutomationEvent(receiver, eventReferenceName)`
- `endNotifyAutomationEvent(receiver, eventReferenceName)`

Both of the methods have the same parameters. The **receiver** parameter indicates which object the event should be delivered to and the **eventReferenceName** parameter indicates which set of events we are interested in.

Event notification can be turned on and off at various times, depending on when you want to receive events. It is also possible to have two or more event references whose type is of the same event interface class, each of which can implement different events.

The diagram in Figure 9 shows the above steps for preparing to receive and process automation events. “Example 3 – Catching Excel Events”, later in this document, takes you through this process in more detail.

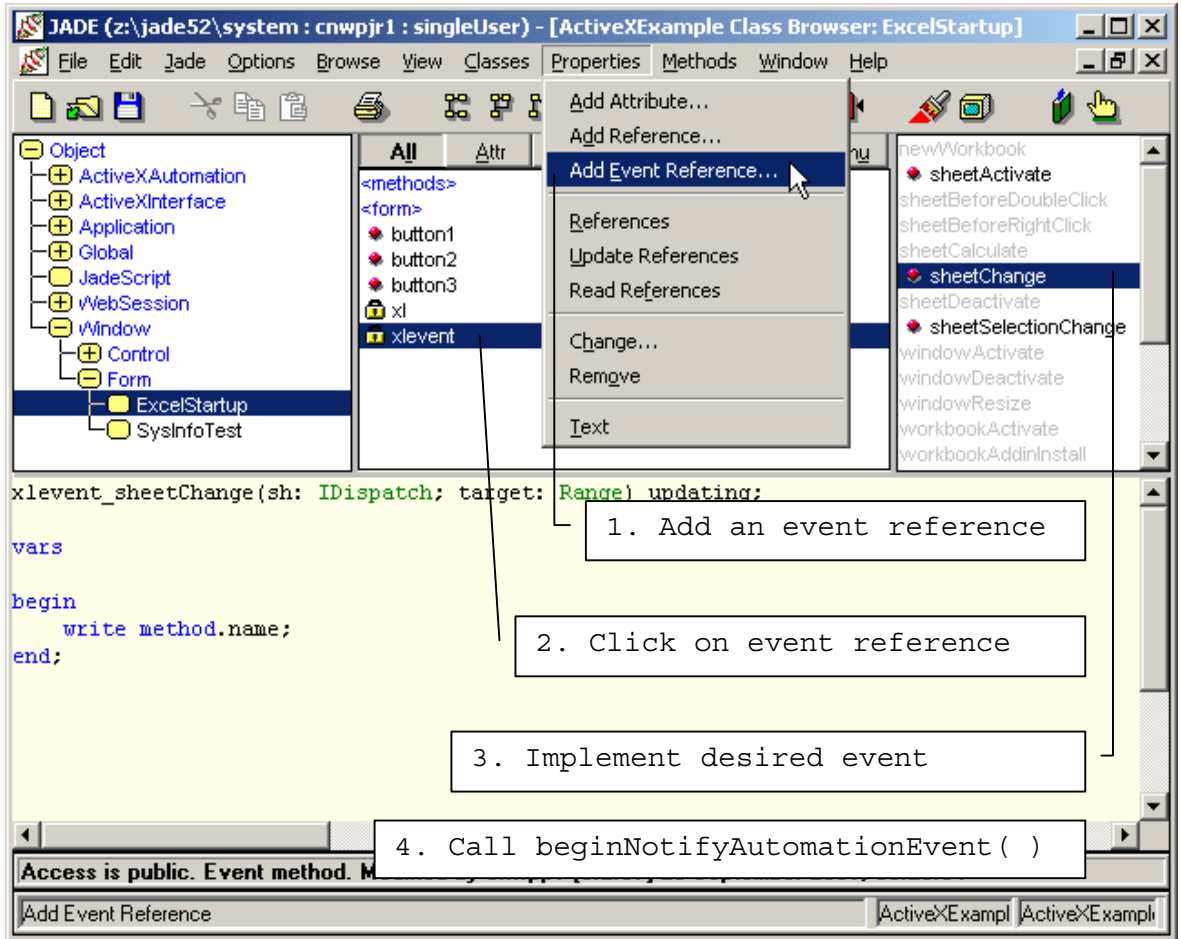


Figure 9 – Steps in setting up for automation event handling

## Optional Method Parameters

COM methods can specify a parameter as being optional. This means that when the method is called, a value for that parameter does not have to be specified. For example, a method may be defined as **mth(param1, param2, param3)** where **param2** and **param3** are optional. Valid calls to the **mth** method would be **mth(p1)**, **mth(p1,p2)**, or **mth(p1,p2,p3)**. Note that the call **mth(p1, p3)** is invalid. When an optional parameter is not specified, no other parameters to the right of this parameter's location can be specified.

JADE does not have a concept of optional parameters. However the JADE interface with COM allows optional parameters to be passed into COM by specifying **null** in the JADE method call that corresponds to an optional parameter. Therefore, using the above method calls from JADE, you could write **mth(p1, null, null)**, **mth(p1,p2,null)**, or **mth(p1,p2,p3)**. The **null** value is used as an indicator of an unspecified optional parameter.

## ActiveX Performance

A common question is “How fast are ActiveX controls”? The answer is “that depends”.

Making a COM call from JADE involves the following steps:

- Input parameters are packaged into a format suitable for COM
- Transmission to the COM server
- Unpacking the parameters
- Invoking the actual method
- Re-packing the return and output parameters
- Transmission back to JADE
- Unpacking the return and output parameters

This all takes time, but a surprisingly large number of methods can be invoked in a short time, particularly when the COM server is *in-process* (the server is a DLL). Even when the server is *out-of-process* (a standalone executable), the throughput is still respectable. The following table shows the number of simple method calls (one input parameter, and a return value) that were achieved on a 400MHz PII computer.

Type of Method Call	Calls Per Second
JADE control method	62,000
ActiveX control method (in-process server)	3,300
Automation method (out-of-process server)	1,400

Table 5 - Method calls per second

3,300 methods calls per second on a control looks good, but compared to method calls on a native JADE controls, it is not so great. So how well do ActiveX controls perform? Let’s take a couple of examples.

Our first example compares the JADE **ProgressBar** control with the same named control in the “Microsoft Windows Common Controls Library”. Both controls show how much of a task has been completed.

In the JADE control, you specify the ‘number of parts’ in the task and then update the ‘parts done’ property as you make your way through the task. The display is updated showing the percentage of the task completed.

The Microsoft control requires a minimum and a maximum to be initialised. A value is then set that represents the progress through the task. Again, the display is updated to show the percentage of the task completed

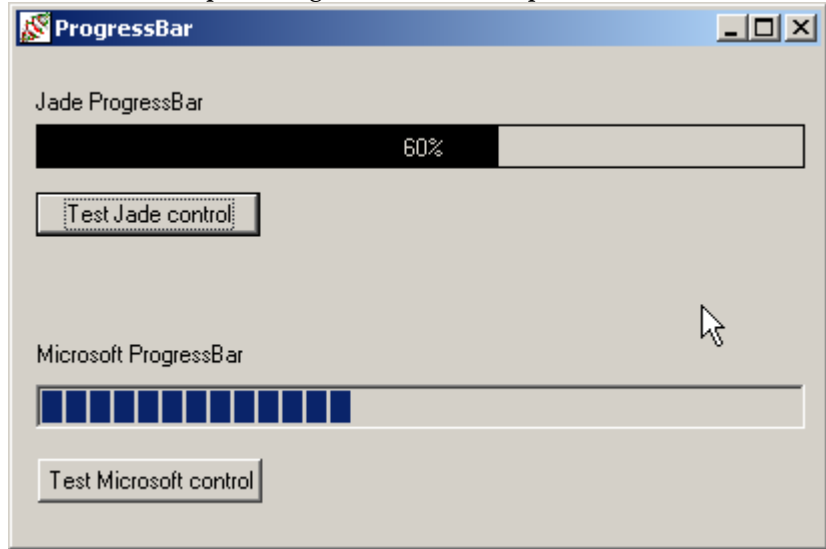


Figure 10– JADE and Microsoft’s ProgressBar controls

Figure 10 shows the test form with the two **ProgressBar** controls on it. Logic was added to increment the ‘parts completed’ in one percent steps from 0% through 100%. This was repeated one thousand times. The average times for this are shown in Table 6.

Control Type	Time for 1000 Loops of 100 Increments
JADE ProgressBar	1733msec
Microsoft ProgressBar	503msec

Table 6 – Time to update ProgressBar controls

This shows that the Microsoft control is three-times faster than JADE, however in real life you are not likely to want to draw the control 1000 times within a few seconds. But what is the reason for the speed difference? The Microsoft control is compiled Windows code in the form of a DLL. The JADE **ProgressBar** control is written in JADE (as user-defined controls would be), and is not part of the JADE executable as the other native JADE controls are.

Our second example involves building a tree-type structure. We will use the Microsoft Common Controls control **TreeView** and compare this with the JADE **ListBox** control. Each control is set up to display a tree where the root node has 10 children, each of these children has 10 children and so on for three levels (a total of 1110 entries).

Figure 11 shows a form with both the JADE and Microsoft controls loaded with the test data and an example node expanded. Table 7 shows the time taken to load the 1,110 entries.

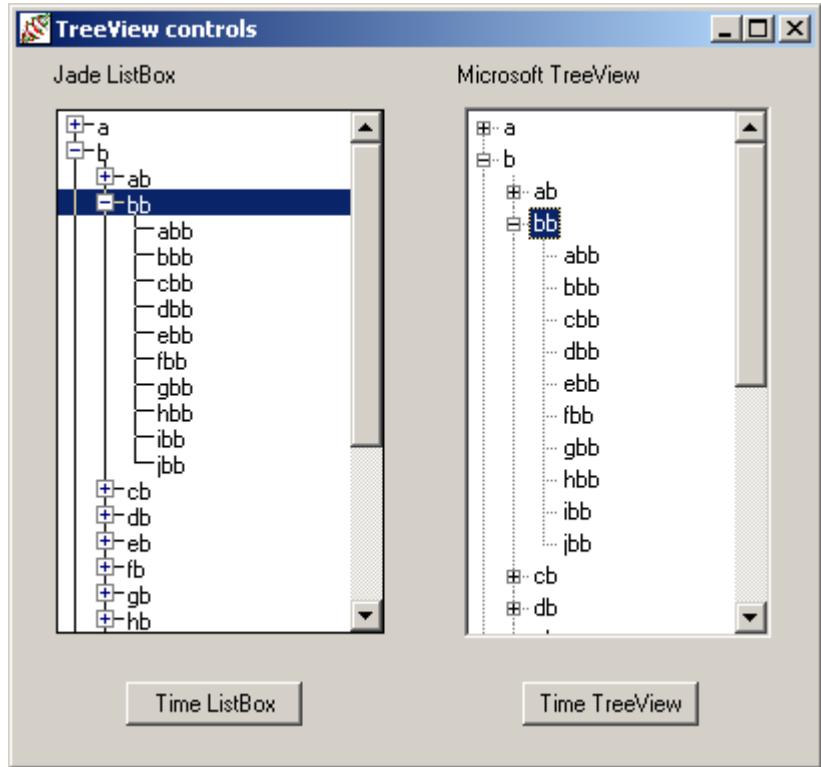


Figure 11 - The JADE ListBox and Microsoft TreeView controls

Control	Load Time
JADE ListBox	812msec
Microsoft TreeView	2504msec

Table 7 - Time taken to load 1110 entries

Again we have a significant difference in speed. This time the **TreeView** control is much slower than the JADE control. Again, why? With a JADE **ListBox control**, you use the **addItem** method to add strings to your **ListBox** and the **itemLevel** array to set the level of the item at a given row. In the **TreeView** control, a node object is created for each entry. On each node you specify the text and level. Each node is a COM object, and therefore returns an interface of type **INode**, which in turn causes a corresponding JADE **INode** instance to be created. In our example above, it means that 1,110 instances of **INode** have to be created within JADE.

Performance of an ActiveX control comes down to how it has been implemented. A control that creates lots of subobjects is likely to be sluggish because each of those subobjects will require the creation of a JADE instance of the interface class that corresponds to that subobject.

## Example 1 – Web Site Searcher

This example is designed to show you how to use ActiveX controls in developing an application. It is not necessarily designed for usefulness.

Figure 12 shows a screen shot of our application. It works in much the same way as the Windows Explorer. On the left side is a tree of page names, while on the right side the actual page is displayed. The root of our display is first entered in the **Start at** text box. Clicking entries in the tree causes that page to be scanned for its links and have those links inserted in the tree list. The clicked page is also shown in the right panel.

This can be achieved with three methods and a total of about 50 lines of JADE logic (excluding comments and blank lines).

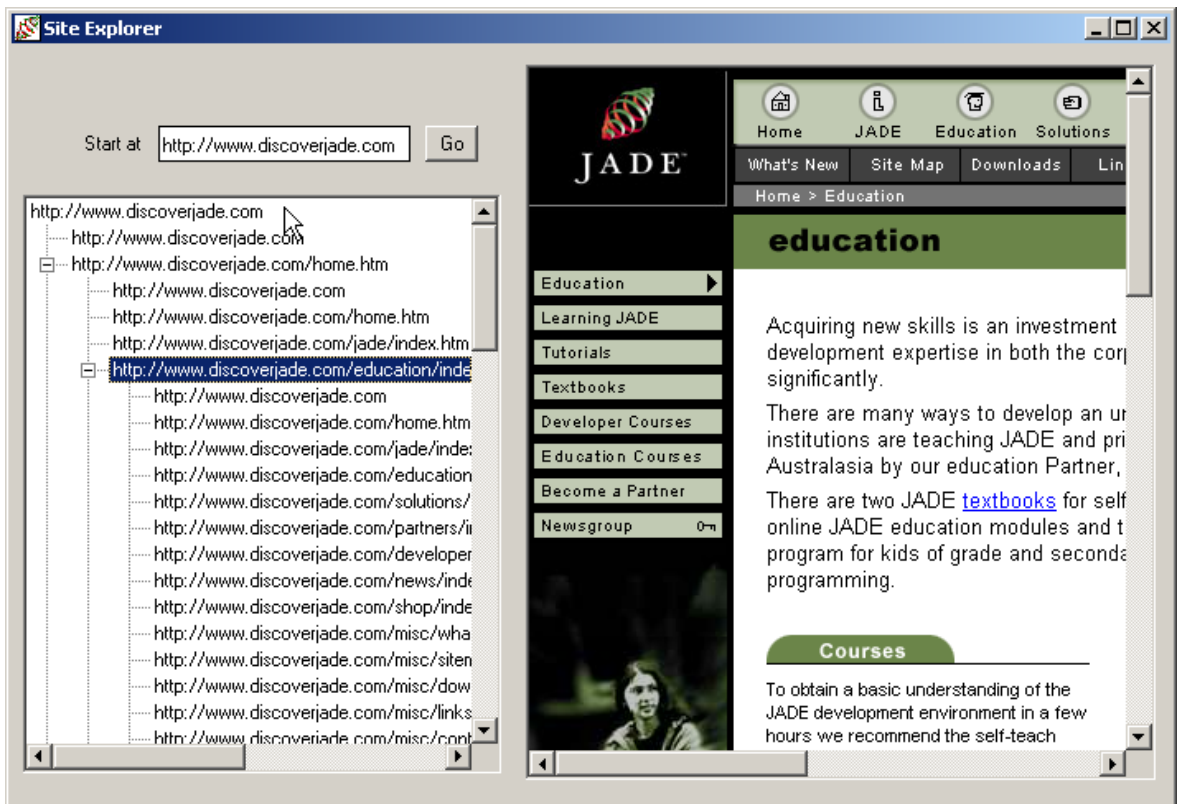


Figure 12– The Web Site Explorer

### How to Use the Web Site Explorer

In the **Start at** text box, enter your starting site address. Pressing the **Go** button adds this site and all of the links on that initial page into the list control. If a link appears twice on a page, it is only shown once.

Clicking on an entry in the tree list:

- Load the page and displays it in the right-hand panel of the form
- Loads all of the links of that page into the tree list as children of the clicked entry

Clicking on the **plus** box then opens the node to show all of that node's links.

## ActiveX Controls Used

The following ActiveX control libraries are used in this example.

- Microsoft Windows Common Controls 6.0
- Microsoft Rich Textbox Control 6.0
- Microsoft Internet Transfer Control 6.0
- Microsoft Internet Controls

These control libraries are installed as part of Windows. If they are not on your machine, they can be downloaded from the Microsoft Web site.

The four ActiveX control libraries listed above need to be imported into JADE. The JADE names offered during the import can all be accepted without modification. However, you may want to make the library names a little more meaningful as they will have been truncated to 30 characters. After importing these controls, they should be visible in the JADE Class Browser. Figure 13 shows the classes generated by the import of these libraries.

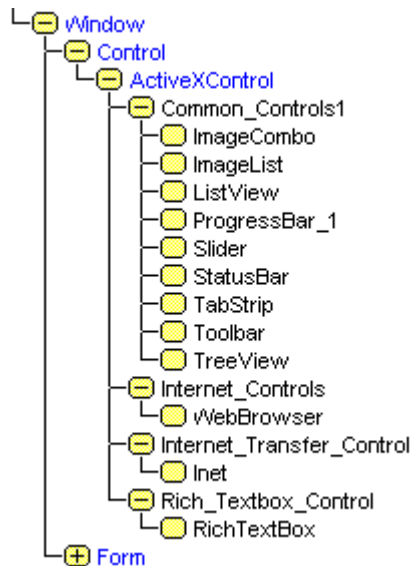


Figure 13– JADE Class Browser showing imported ActiveX controls

## Creating the Form

Use the JADE Painter to create a form as shown in Figure 14. Make it a fairly large form, as the **WebBrowser** control is used to display actual Web pages.

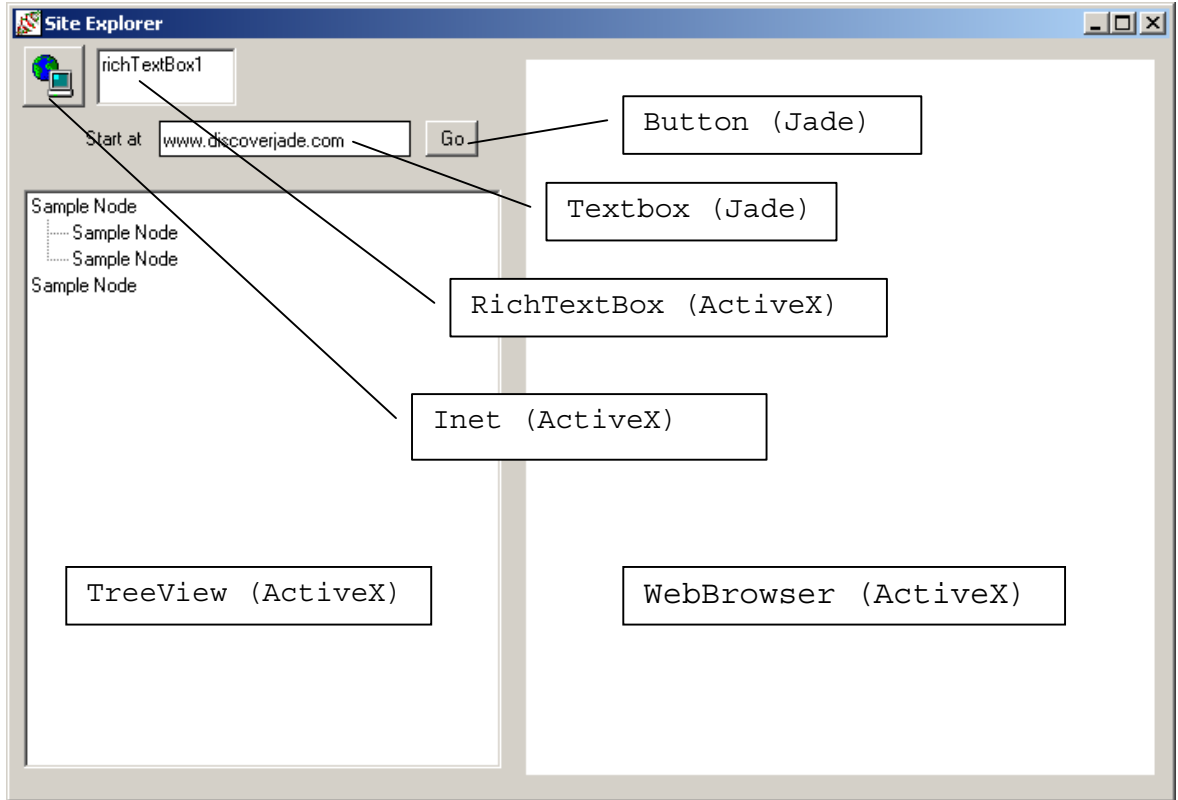


Figure 14 – Form layout for Site Explorer

Out of all the controls, there is only one property you need to change and two others you might like to adjust via the Painter's Properties dialog. For the **RichTextbox** control you should set the **visible** property to **false**. On the **Specific** sheet of the Properties dialog for the **TreeView** control, the property **indentation** can have its value reduced to about 25 (from 39). The caption on the button can be changed to something more meaningful.

Notice the difference between the runtime layout and the Painter layout. We have set the visible property of the **RichTextbox** to **false** to hide this at run time. The **Inet** control automatically disappears at run time as it has no user interface.

## The TreeView Control

Each entry in the **TreeView** control is a *node*. When adding a node you can specify its relationship to other nodes and hence build up a tree structure. Each node is a COM object and has an interface to access it by. In JADE, the node interface was imported as **INode**. The **TreeView** control has a property **nodes** which is a collection of node objects. This collection is also a COM object with an interface, in this case called **INodes**.

To add a row to the **TreeView** control requires us to add a node to the nodes collection. The **INodes** interface provides the **add** method for creating and adding a new node to the collection. Therefore the JADE logic to add a new row to the **TreeView** control would look something like:

```
treeView1.nodes.add(parentNode, TreeView.TvwChild, null,
                    EntryText, null, null);
```

The **INodes::add** method returns an **INode**. This interface can then be used to set properties of the node (such as font, colour, and so on).

## The Inet (Internet Transfer) Control

The Internet transfer control allows you to access a site and retrieve files using **ftp** or **http**. In our example it is used to import the HTML pages and load the text into the **RichTextbox** control for further processing.

The only method we use of this control is **openURL**.

## The RichTextbox Control

This control is similar to the standard textbox but provides more functionality in such areas as formatting. In our case, the only method we are interested in is the **find** method.

We use this method to scan our imported page for links to other pages.

## The Supporting Logic

The user adds their starting URL in the **Start at** text box and presses the **Go** button. The button's click method then:

- Adds the root site to the tree list
- Calls the method **addLinks** to add the links found on that page to the **TreeView** list
- Expands the parent node

The logic to perform this task follows:

```
button1_click(btn: Button input) updating;

vars
    rootNode : INode;
begin
    app.mousePointer := MousePointer_HourGlass;

    // Re-initialize, in case of multiple runs
    treeView1.nodes.clear;
    if not textBox1.text [1:7] = "http://" then
        textBox1.text := "http://" & textBox1.text;
    endif;

    // Load start node
```

```

        rootNode := treeView1.nodes.add(null,
                                        TreeView.TvwChild,
                                        null,
                                        textBox1.text,
                                        null,
                                        null);
    webBrowser1.navigate(textBox1.text,
                        null,
                        null,
                        null,
                        null);
    addLinks(rootNode);
    rootNode.expanded := true;

    app.mousePointer := MousePointer_Default;
end;

```

Figure 15 – Source of the Go button's click method

Note how the expanding of a node in the **TreeView** control is done via a method of the node and not the **TreeView** control. Compare this to how you would expand a node for a **JADE ListBox**.

The **addLinks** method uses the **Inet** control to load the source for the page into the **RichTextBox** control. The **find** method of the **RichTextBox** control is used to scan for links embedded within that page. Any links found are added to the **TreeView** control.

```

addLinks(parentNode: INode);

vars
    startIndex, endIndex : Integer;
    lnk : String;
    links : HugeStringArray;

begin

    create links transient;

    // Load the page into a RichText control
    richTextBox1.text := inet1.openURL(parentNode.text,
                                       null).String;

    // Scan page for any links
    startIndex := richTextBox1.find('http://',
                                    0,
                                    null,
                                    null);

    while (startIndex > 0) do
        // We have found the start of a URL,
        // so extract the full name
        endIndex := richTextBox1.find(' ',
                                      startIndex+1,
                                      null,
                                      null);
        richTextBox1.selStart := startIndex + 1;
        richTextBox1.selLength := endIndex - startIndex-1;
        lnk := richTextBox1.selText;
    end;

```

```

// If this is the 1st time we have found this URL
// on this page then add it to the list of links.
if not links.includes(lnk) then
    treeView1.nodes.add(parentNode,
                        TreeView.TvwChild,
                        null,
                        lnk,
                        null,
                        null);
    links.add(lnk);
endif;

// Scan for next URL
startIndex := richTextBox1.find("http://",
                                endIndex + 1,
                                null,
                                null);

endwhile;

epilog
    delete links;
end;

```

Figure 16 – JADE source for the addLinks method

When the user clicks on a node in the **TreeView** control, a **nodeClicked** event is triggered. On receipt of the **nodeClicked** event two things happen:

- The page is displayed in the WebBrowser control
- The **addLinks** method is called to load this page's links into the TreeView control if these links are not already loaded (that is, we don't want duplicate entries)

```

treeView1_nodeClick(cntrl:TreeView input; node_:INode)
                                updating;

vars

begin
    app.mousePointer := MousePointer_HourGlass;

    webBrowser1.navigate(node_.text,
                        null,
                        null,
                        null,
                        null);

    if node_.children = 0 then
        addLinks(node_);
    endif;

    app.mousePointer := MousePointer_Default;
end;

```

Figure 17 – treeView1\_nodeClick event method

## Example 2 – Graphing with Excel

Having decided that we will use Excel to generate a chart, how do we go about this using JADE automation?

The first step is to import the Excel automation type library. Importing an automation library was covered at the start of this article. In all but two cases, we will accept the JADE names offered. The exceptions are:

- On the Import Wizard's page 2, enter **Excel** for the Library name.
- On page 3, enter **ExcelApp** for the name of the application object. (JADE offers the name **Application\_1** to avoid conflict with the JADE RootSchema class **Application**.)

The import of the type library is likely to take minutes. Excel is a very large product having close to 200 interfaces and thousands of methods. Once the import is completed, you should have a structure similar to that in Figure 18.

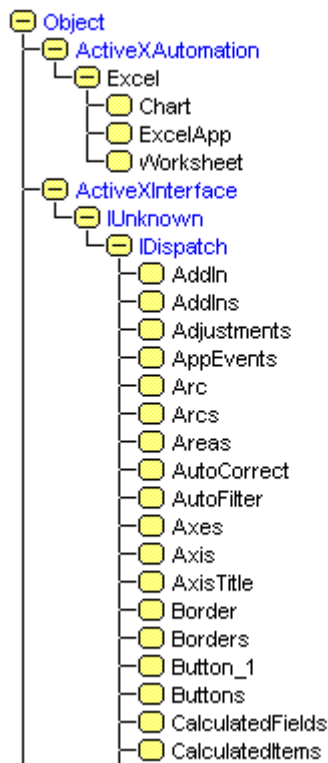


Figure 18 - JADE hierarchy after Excel import

The first task is to create a JADE object corresponding to the Excel application. This is done via the standard JADE **create** command, as follows.

```
create xl;
```

Note that this only creates a JADE instance of the **ExcelApp** class and no COM object is created yet.

Before the COM object is created, we can specify where the COM server is to run. In our example, we want it to run on the JADE application server machine when in thin client mode. This is done by:

```
xl.usePresentationClient := false;
```

We could also specify a machine by setting the **remoteServerName** property.

We are now ready to create the COM object. This is done in one of two ways: by accessing a property or method, or by calling the **createAutomationObject** method, as follows:

```
xl.createAutomationObject();
```

Normally we would not want the Excel application window to be visible. However, in this case we want to see Excel at work during the automation process. Excel provides a **visible** property that can be set, depending on whether Excel is to be visible or not, as follows:

```
xl.visible := true;
```

An Excel application is made up of a number of *workbooks*, each with a variable number of *sheets*. The application object provides a **workbooks** collection property that we can add to. The **add** method of the **Workbooks** interface takes a parameter that allows us to specify the type of workbook added. It can be a constant value that indicates the type of worksheet added. The constant values, like all constants, are imported during the import and are defined as JADE class constants on the class that corresponds to the COM library, in this case Excel, as shown in Figure 18. To add a workbook with one sheet to Excel then we use:

```
xl.workbooks.add(Excel.XlWorksheet);
```

To access cells in an Excel application, you must specify a *range* of cells you are working with. The range is specified on a per-sheet basis, so this requires a reference to the sheet. Excel provides an **activeSheet** property to return the current active sheet. This property can return a number of different types (a chart is just one type of sheet). The Excel type library specified **activeSheet** as a general interface and not as an interface to a worksheet. Therefore when imported by JADE, **activeSheet** was given the type **IDispatch** (and not, for example, **I\_Worksheet**). If we have a worksheet and want to work with methods and properties of **I\_Worksheet**'s, we need to cast the returned **activeSheet** value into an **I\_Worksheet**, as follows

```
sht := xl.activeSheet.I_Worksheet;
```

We can now define a range using the **range** method. The **range** method (in JADE) is defined as taking two parameters, the second being optional. Normally the first parameter would be specified as a string, representing the cells selected; for example, **range**("A1", **null**), or **range**("A1:B2", **null**). Alternatively, a range of cells can be specified by the first parameter being set to the top-left cell in the range and the second parameter to the bottom-right cell in the range (for example **range**("A1", "B2")). In the first case, notice the use of **null**. COM defines the second parameter of the **range** method as being optional. JADE has no concept of optional parameters. If you don't want to pass a value via an optional parameter, set that parameter to **null**. The first cell (column A, row 1) is selected by:

```
sht.range("A1", null);
```

With a range (or cell selection) defined, we can now access individual or groups of cells. The **Range** interface provides the **value** property for this. However, in the Excel type library, **value** is defined as a variant (equivalent to a JADE **Any**). JADE does not support properties of type **Any**, so the import is forced to generate a pair of get and put methods. For the **value** property on range, we get the **getValue** and a **putValue** methods generated. In our example, we append this method to the previous statement to get:

```
sht.range("A1", null).putValue("One");
```

This is repeated for all cells to be initialized.

All cells are then selected for the chart building process, as follows:

```
rng := xl.range("A1", "C3");
```

We are now ready to produce our chart. In the same way we added a sheet to the Excel application we can add a chart, as follows:

```
chrts := xl.charts;
chrts.add(null, null, null, null);
chrt := xl.activeChart;
```

When using Excel, you would probably use the chart wizard tool to help you build the chart. Well, you can do the same thing when automating.

```
chrt.chartWizard( rng,           //source
                  Excel.Xl3DPie, //gallery
                  7,             //format
                  Excel.XlRows,  //plotBy
                  1,             //categoryLabels
                  0,             //seriesLabels
                  true,          //hasLegend
                  "Our Example", //title
                  null,         //categoryTitle
                  null,         //valueTitle
                  null);        //xtraTitle
```

Individual parameters of the **chartWizard** method are not discussed here. However it is worth noting the use of constant values (**Excel.Xl3Dpie**, **Excel.XlRows**, and so on) and the **null** values passed to the last three optional parameters. If experimenting with this code, check the Excel documentation for valid values for the **format** parameter as this depends on the value of the **gallery** parameter.

We can now print our chart. The **printOut** method does this. All parameters are optional, so we can specify **null** for all parameters. Note that if you specify **null** for an optional parameter, all other optional parameters to the right of that parameter must also be **null**. Our command to print is therefore:

```
chrt.printOut(    null,          //1st page
                 null,          //last page
                 null,          //copies
                 null,          //preview
                 null,          //printer
                 null,          //print to file
                 null);        //collate
```

Our last task is to close Excel and delete our JADE transient **ExcelApp** object. If you close Excel and have changed a workbook, Excel prompts you to see if you want that workbook saved. In our case, we don't want to save the data or be prompted to save. The way around this is to mark the workbook (our data sheet and chart sheet) as having been saved. Excel can then be closed via the **quit** method. The JADE Excel object is deleted with a **delete** command. So our final clean-up code looks like the following:

```
xl.activeWorkbook.saved := true;
xl.quit;
delete xl;
```

To finish, the method we have been building is presented here as a complete **JadeScript** method.

```
xlChartExample();

vars
  xl:ExcelApp;
  sht:I_Worksheet;
  rng:Range;
  chrts:Sheets;
  chrt:I_Chart;

begin
  // Create JADE and COM objects, run COM server
  // on App Server if in thin client mode
  create xl;
  xl.usePresentationClient := false;
  xl.createAutomationObject;

  // Make Excel visible so we can see what is happening

  xl.visible := true;

  // Add Workbook (with one sheet)
  xl.workbooks.add(Excel.XlWorksheet);
```

```

// Fill out cells of top sheet
sht := xl.activeSheet.I_Worksheet;
sht.range("A1", null).putValue("One");
sht.range("B1", null).putValue("Two");
sht.range("C1", null).putValue("Three");
sht.range("A2", null).putValue(10);
sht.range("B2", null).putValue(5);
sht.range("C2", null).putValue(3);

// Create a chart
chrts := xl.charts;
chrts.add(null, null, null, null);
chrt := xl.activeChart;

// Select the range of select we are going to chart
rng := sht.range("A1", "C2");

// Use Excel "chart wizard" to generate chart
chrt.chartWizard( rng, //source
                  Excel.Xl3DPie, //gallery
                  7, //format
                  Excel.XlRows, //plotBy
                  1, //categoryLabels
                  0, //seriesLabels
                  true, //hasLegend
                  "Our Example", //title
                  null, //categoryTitle
                  null, //valueTitle
                  null); //xtraTitle

// Output chart
chrt.printOut( null, //1st page
              null, //last page
              null, //copies
              null, //preview
              null, //printer
              null, //print to file
              null); //collate

epilog
  if xl <> null then
    // Mark workbook as saved, so we don't get
    // ask "do you want to save?" on exit
    xl.activeWorkbook.saved := true;

    // Close down Excel
    xl.quit;

    // Delete the JADE object
    delete xl;
  endif;
end;

```

Figure 19 – JADE script to produce Excel chart

The biggest problem with products such as Excel is knowing what methods and objects to use to achieve a given task. The Excel help file is a valuable resource. Unfortunately this is written from a Visual Basic user's point of view. However in most cases the transformation from a VB example into JADE code is fairly straightforward.

One way to find your way around the object model of Excel (and other Microsoft Office applications as well) is to use the macro recorder. From the Excel tools menu, choose **Macro\Record New Macro** and then manually perform the task you want to automate. Once your task is completed, choose **Macro\Stop Recording** from the tools menu. Again, from the tools menu choose **Macro\Macros...**, select the macro you have just created, and click **edit**. You then see generated Visual Basic code that perform the task you have just completed. You will to translate the Visual Basic code into JADE, but at least you know what methods need to be called.

## Example 3 - Catching Excel Events

---

In this example we show you how to catch events fired by automation objects.

Our task is to color the cells of an Excel spreadsheet depending on the data entered. Negative numeric data is colored red, a zero or a positive valued cell is colored blue, and all other cells (text) are colored green. When not catching events, the cell colour is the default color (the actual colour depends on your system settings but is probably black).

Before proceeding with this example, you should make sure you understand the workings of “Example 2 – Graphing with Excel”. Our discussion here assumes that the Excel library has already been successfully imported, as discussed in that example.

Create a form with two command buttons and a checkbox on it as shown t in Figure 20. Each control’s caption has been made the same as its name. The **startExcel** and **exitExcel** buttons start and exit Excel, respectively. The **colourCells** check box will tell Excel when we require notification of cell change events so we can perform our coloring.

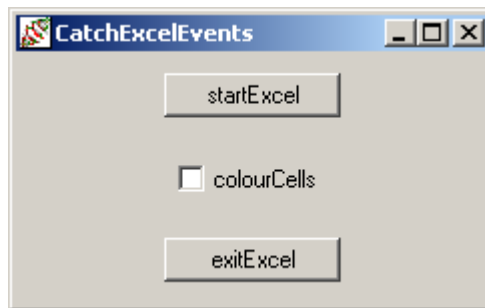


Figure 20 – Screen layout of CatchExcelEvents example

Open the JADE Class Browser with the **Form** class selected. Add a reference property of type **ExcelApp** to our form class, giving it the name **xl**. This lets us store a reference to our automation object (Excel). If you imported Excel using a different name for the Excel application object, use that instead of **ExcelApp**.

Figure 21 shows the JADE Class Browser with our form class displayed after adding a reference to **ExcelApp**.

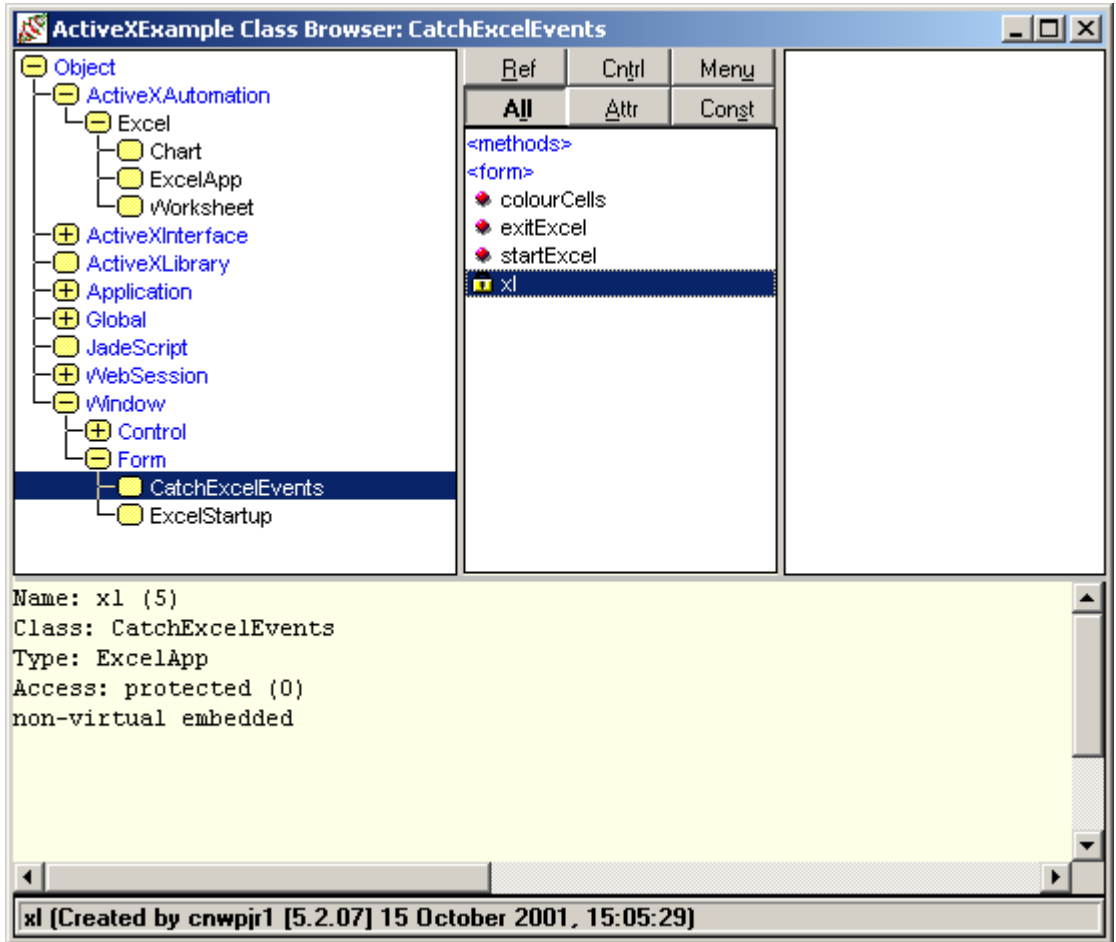


Figure 21 – CatchExcelEvent form class

We will create our instance of the Excel object when the **startExcel** button is clicked. In the **click** method of the **startExcel** button, add the following code:

```
begin
  create xl;
  xl.usePresentationClient := false;
  xl.createAutomationObject();
  xl.visible := true;
end;
```

Those of you who have worked through “Example 2 – Graphing with Excel”, will be familiar with the above code. The setting of the **visible** property to **true** is important. This makes Excel visible, which it has to be if we are to enter cell data.

The **click** method of the **exitExcel** button provides the Excel close down code. Again this will be familiar to those who have studied the Excel graphing example.

```
begin
    xl.activeWorkbook.saved := true;
    xl.quit;
    delete xl;
end;
```

When we click on a control reference, the Class Browser displays a list of events that we can implement in the methods list pane of the browser window. We can then add logic to these events that will be executed when the event is triggered. We have just done this by adding **click** events to our **startExcel** and **exitExcel** buttons. Automation events work in a similar way. First we must add a reference to an event interface. This is done via the **Add Event Reference** option of the Properties menu shown in the JADE Class Browser, as shown in Figure 22.

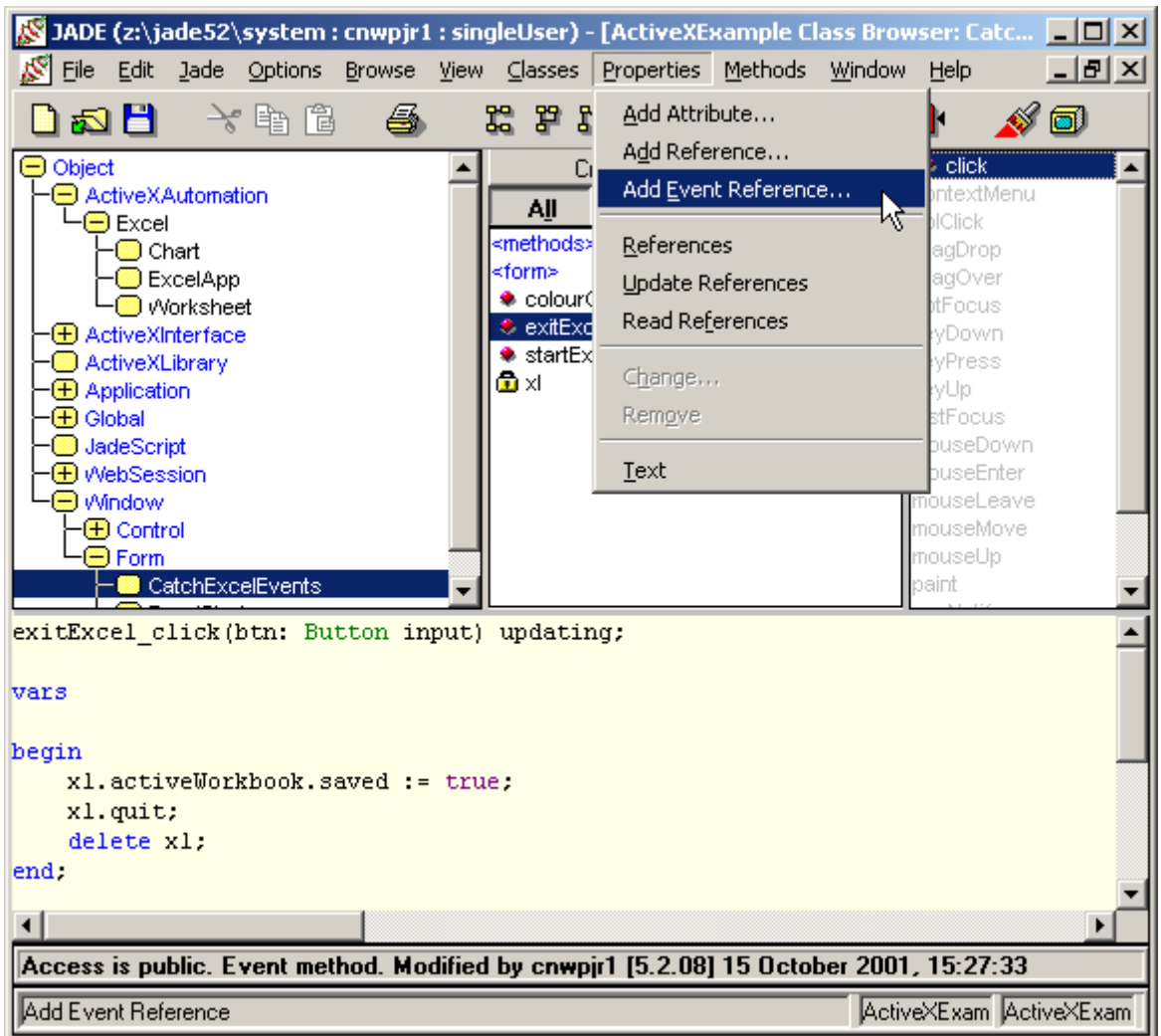


Figure 22 – Adding an event reference

The **Add Event Reference** menu option brings up the dialog shown in Figure 23. In this case, enter **xlEvent** for the name and select **AppEvents** from the list of all automation event interfaces available in this schema. Automation event classes are **IDispatch** subclasses used to implement event handling. If you refer back to Figure 18 of “Example 2 - Graphing with Excel”, you will see **AppEvents** is listed as a subclass of **IDispatch**.

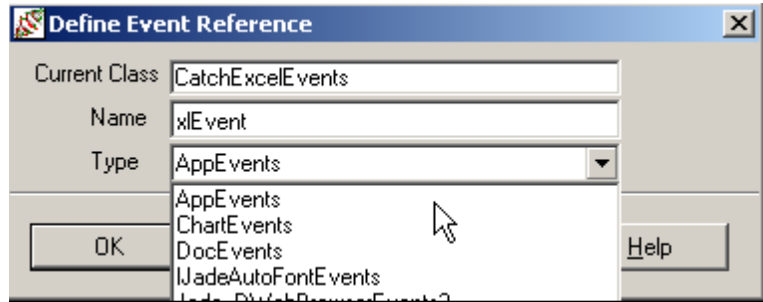


Figure 23 – Defining an event reference

When the event reference property has been added, selecting it in the Class Browser displays a list of event methods that can now be implemented. Figure 24 shows the JADE Class Browser with the event reference selected.

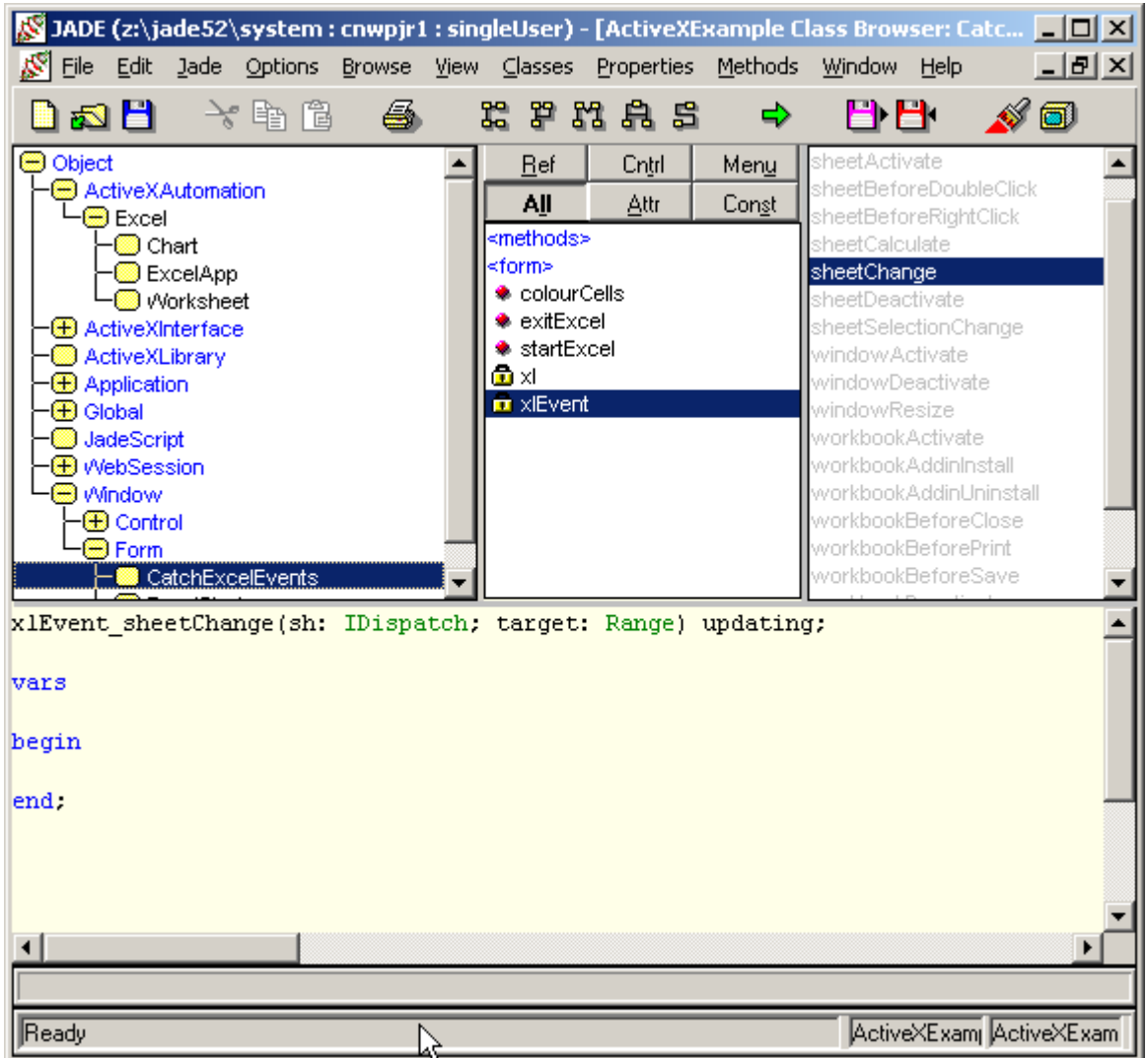


Figure 24 – A selected Automation Event Reference

Notice the list of event names in the methods list pane. As with a control, the methods are greyed; that is, they are unimplemented.

Select the **sheetChange** method and add the following code. This event is fired whenever something changes on a sheet and is not an indication of the user changing (or swapping between) sheets.

```
vars
    cell : Range;

begin
    cell:=xl.activeCell;

    // Colour the cell
    if xl.worksheetFunction.isNumber(cell) then
        if cell.text.Integer < 0 then
            cell.font.putColor(Red);
        else
            cell.font.putColor(Blue);
        endif;
    else
        cell.font.putColor(Green);
    endif;

end;
```

The Excel object's default interface provides us with a number of useful functions. Here we get the current active cell by accessing the **activeCell** property. The Excel object also provides the **worksheetFunction** property. This is a read-only property that provides us with the interface **WorksheetFunction**. This interface has many useful methods including the one used here, **isNumber**. The method **isNumber** returns **true** or **false** depending on whether the cell contains a numeric value. Notice how we access the cell's font color by going via the font method (that returns a font interface) to access the **putColor** method.

Everything has now been set up. However, by default, automation does not pass on any event. We control the receipt of event notifications by the use of two methods, as follows:

- `beginNotifyAutomationEvent(receiver, eventReferenceName)`
- `endNotifyAutomationEvent(receiver, eventReferenceName)`

In our example, depending on the state of the **colourCells** check box, we will turn on and off automation event handling by calling these methods. The **receiver** parameter indicates which object the event notification should be delivered to, in our case, the form. The **eventReferenceName** parameter is set to the name of the automation event reference parameter that implements the events we are interested in.

Add the following code to the **click** event of the **colourCells** check box.

```
begin
    // Turn on/off event notification depending
    // on whether we want to do cell colouring
    if colourCells.value = true then
        xl.beginNotifyAutomationEvent (self, "xlEvent");
    else
        xl.endNotifyAutomationEvent (self, "xlEvent");
    endif;
end;
```

Try it out. Start Excel and enter cell data. This data is in the default font color. Now go back to the JADE application and click the **colourCells** check box. Return to Excel and enter more data. This time the cell should be colored depending on data entered (red for negative numbers, blue for positive, and green for non-numeric cells). If you return to JADE and once again click the **colourCells** check box to end event notification, data entered into Excel is now left in the default color. Note that cells already colored, will (once notification is turned off) remain in that colour.