




ActiveX Controls and Citect SCADA

By Warwick Black

ActiveX Controls	2
Overview	2
Figure 1: The Visual Interface, Properties, Methods and Events allow CitectSCADA to interact with the ActiveX control.ActiveX Control Members:	2
ActiveX Control Members:.....	3
Properties: 	3
Methods: 	3
Events: 	3
Creating ActiveX Object Instances in Citect SCADA	4
Creating instance during Design-Time	4
Creating Control Object instance during Run-Time:.....	6
Creating an Object instance during Run-Time (i.e non Visual):	7
Using ActiveX Objects in Citect SCADA.....	8
Linking Properties to SCADA Tags in Design-Time	8
Using ActiveX Controls via CiCode.....	10
Referencing an ActiveX Instance	10
Using Properties from CiCode.....	11
Using Methods from CiCode	12
Writing Event Handlers in CiCode:.....	13
Advanced Concepts	14
Accessing Properties with Indexes	14
Accessing members of 'Nested' Classes	14
iDispatch Interface and Citect Compatability.....	15
Linking Properties to SCADA Tags at Runtime via CiCode	16
ActiveX Cicode Function List	17
Link / Recommended Reading.....	18

ActiveX Controls

Overview

Active X Controls (formerly called OLE Controls) are ready-made components which can be added to a 'Container Program', such as CitectSCADA, to add its functionality to that program.

The ActiveX could be considered analogous to an 'Include Project' used to add a library of pre-defined functions and sometimes a graphical component to the SCADA project.

ActiveX objects with a visual interface are commonly referred to as an 'ActiveX Control Object', and those without a visual interface, simply an 'ActiveX Object'.

Unless you have created the ActiveX yourself (not covered in this document) we are not concerned with what happens inside the ActiveX itself. instead we are just concerned with what interfaces that the Object exposes to Citect SCADA.

There are 3 types of interfaces or 'Members' to an ActiveX Control that we are interested in.

These are 'Properties', 'Methods' and 'Events' and are explained in more detail in the next section.

The easiest way to view all Interfaces or 'Members' of an ActiveX control is to use the VB Object Explorer, instructions on how to do this with MS Excel are laid out in Knowledgebase (KB) article Q4102.

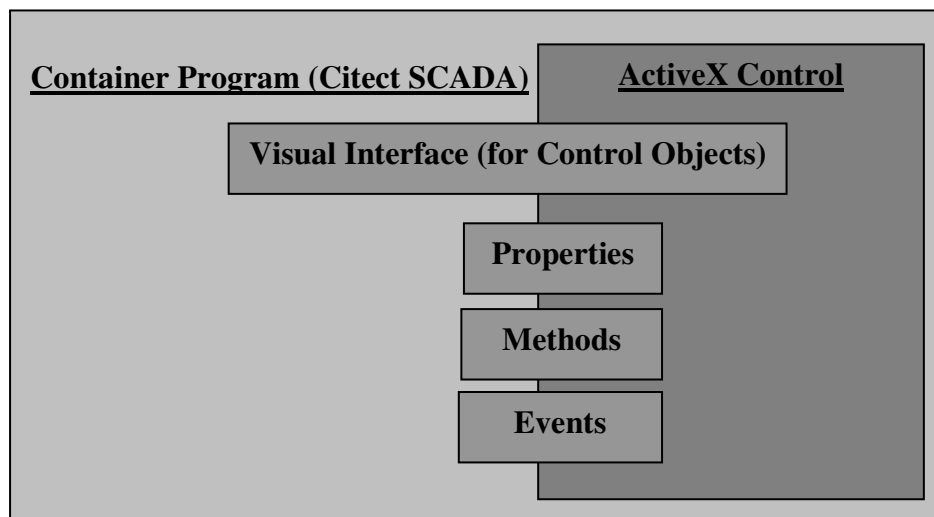


Figure 1: The Visual Interface, Properties, Methods and Events allow CitectSCADA to interact with the ActiveX control.

ActiveX Control Members:

The following screenshot, obtained following KB article Q4102, shows some of the members available inside the MS Forms 'TextBox' ActiveX control.



Properties:

Properties are variables that determine how a control will behave. These are either set at Design Time, during Runtime via CiCode, or internally by the ActiveX itself.

In the above example, the property, MultiLine, can be set to True or False, and controls whether or not to Text is confined to one line only or not.

Likewise, the property, Scrollbars, can be set to True or False, and controls whether or not to display Scrollbars.

As discussed later in this article, these can be 'Linked' to CitectSCADA Variable Tags during Design-Time, or they can be checked/set via CiCode/CiVBA at Run-Time.

Methods:

A 'Method' of an ActiveX control is an externally accessible function, usually performing an operation on or within the ActiveX control it belongs to.

For example, in the screenshot at the top of the page, we can see that the TextBox control contains the 'Paste' Method (amongst others). This 'Paste' function can now be called from CiCode or CiVBA, during Runtime, to paste the Clipboard contents to that instance of the TextBox Control.

Events:

Events are controlled by the ActiveX itself, and are often used to let the Container Program (i.e CitectSCADA) know when a certain condition has been met.

In the above example, we can see that there are various mouse-related events inside the TextBox control, such as MouseDown. This event will be triggered by the control when it detects a Mouse Down action has been performed on the TextBox.

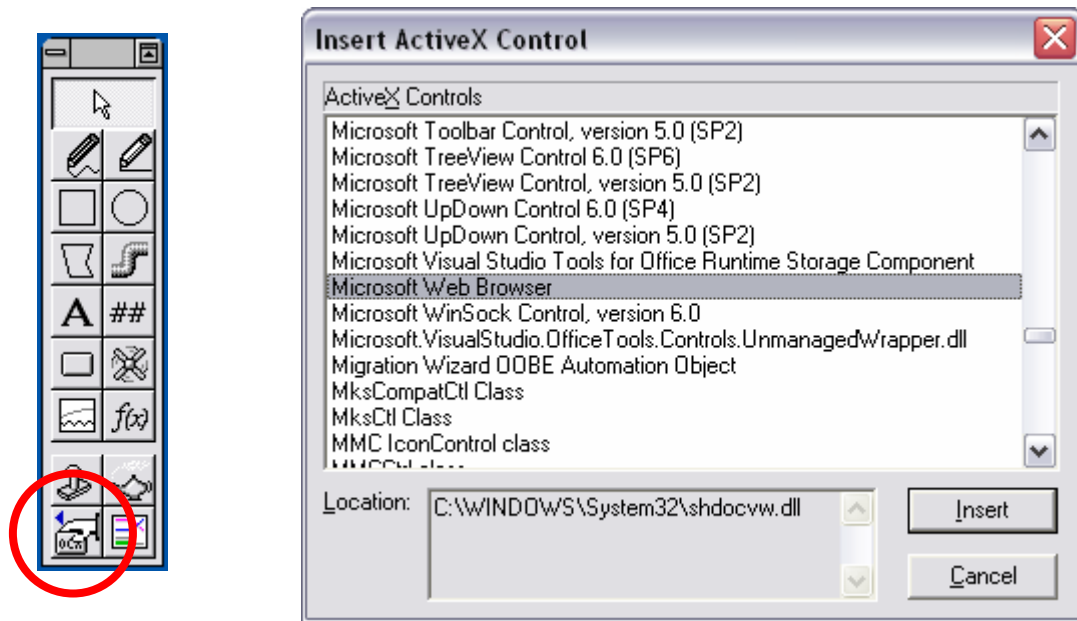
If we want to utilise these events, we need to write an 'Event-Handler' in CiCode or CiVBA, otherwise these operations will be ignored. Writing 'Event Handlers' is discussed later.

Creating ActiveX Object Instances in Citect SCADA

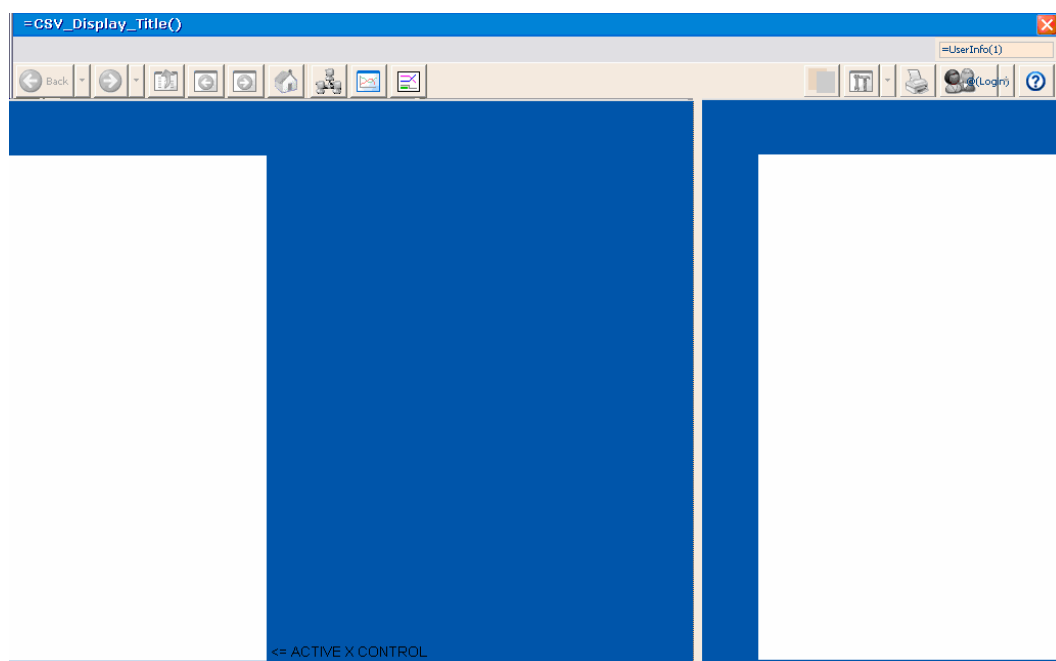
Before using an ActiveX object, it must be added to the CitectSCADA page. This can be done graphically during Design-Time or via code at Runtime. Since you can have multiple sessions of an ActiveX object each one is generally referred to as an 'Instance'.

Creating an instance during Design-Time

In Graphics Builder open a new Page and paste an Active X control onto the Page. For this example we will be using the 'Microsoft Web Browser' ActiveX Control.



Resize the ActiveX control to suit the desired layout, and you page should look as follows:

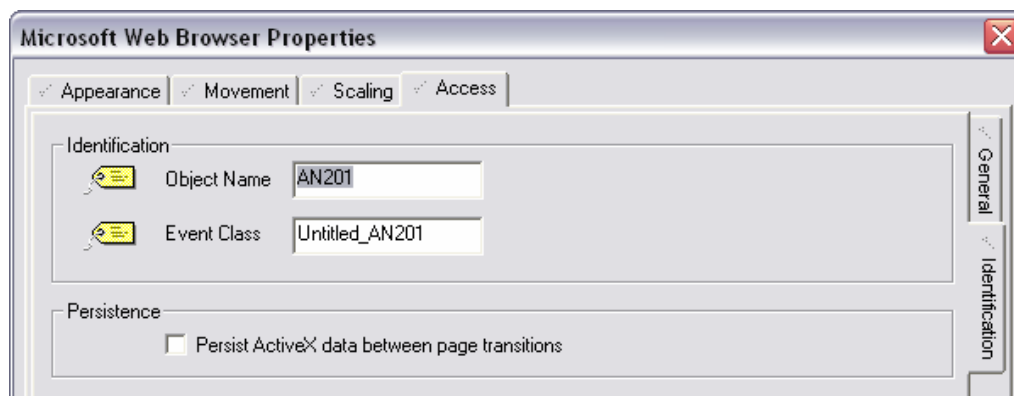


In order to access this object programmatically, we need to know what the 'Object Name' and 'Event Class' of this new instance are.

These are automatically created by CitectSCADA, and can be graphically checked by:

- Right Click the ActiveX instance, Select 'Properties'
- Browse to the 'Access' Top Tab
- Select the 'Identification' Side Tab.

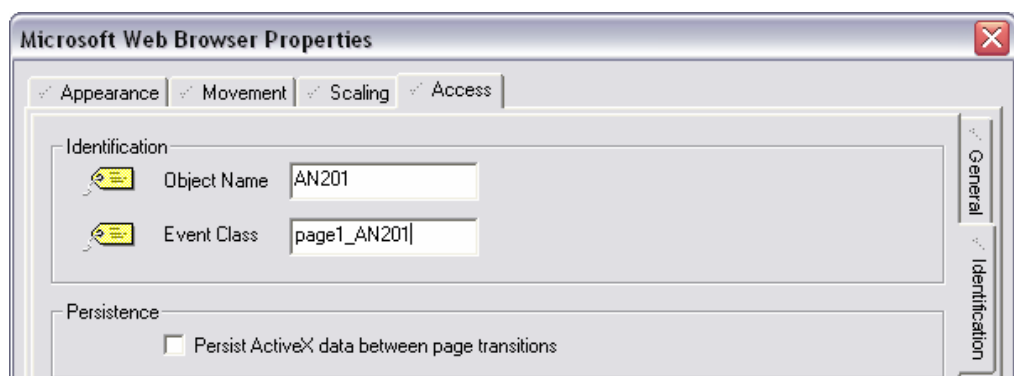
On this screen, we make note of the '**Object Name**' (AN201) and the '**Event Class**' (Untitled_AN201').



The 'Event Class' is automatically generated by the first 8 characters of the Page Name, followed by an underscore and the 'Object Name'.

The 'Object Name' is automatically generated based upon the Animation Number of the object.

If we now save our page, the 'Event Class' has now changed to 'Page1_AN201'.



NOTE:

Note the "Persist ActiveX data between page transitions" option. If checked this will allow your object to retain values whilst changing CitectSCADA pages, so when you come back the values should still be there.

Creating Control Object instance during Runtime:

To create an instance of an 'ActiveX Control Object' you can use the CiCode function: CreateControlObject.

The main difference between this and the Graphical method, is that you have to manually assign the 'Object Name' and 'Event Class' that is created automatically via the Graphical method.

An object created using this function remains in existence until the page is closed or the associated CiCode Object is deleted. This function does not require an existing animation point. When the object is created, an animation point is created internally. This animation point is freed when the object is destroyed.

Syntax

CreateControlObject(*sClass*, *sName*, *x1*, *y1*, *x2*, *y2*, *sEventClass*)

sClass:

The class of the object. You can use the object's human readable name, its program ID, or its GUID. If the class does not exist, the function will fail.

For example:

- * "Calendar Control 8.0" - human readable name
- * "MSCAL.Calendar.7" - Program ID
- * "{8E27C92B-1264-101C-8A2F-040224009C02}" - GUID

sName:

The name for the object in the form of "AN" followed by its AN number, eg. "AN35". This name is used to access the object.

x1:

The x coordinate of the object's top left hand corner as it will appear in your CitectSCADA window.

y1:

The y coordinate of the object's top left hand corner as it will appear in your CitectSCADA window.

x2:

The x coordinate of the object's bottom right hand corner as it will appear in your CitectSCADA window.

y2:

The y coordinate of the object's bottom right hand corner as it will appear in your CitectSCADA window.

sEventClass:

The string you would like to use as the event class for the object.

Return Value:

The newly created object, if successful, otherwise an [error](#) is generated.

Creating an Object instance during Run-Time (i.e non Visual):

To create a non-visual instance of a control object, the CiCode function CreateObject can be used.

This does not create a Visual component, nor does it require an 'Event Class' or 'Object Name' the Return Value of the function provides the handle to refer to this instance programmatically.

CreateObject

Creates a new instance of an ActiveX object. If you use this function to create an ActiveX object, it will have no visual component (only the automation component will be created).

If you assign an object created with the CreateObject() function to a local variable, that object will remain in existence until the variable it is assigned to goes out of scope. This means that such an object will only be released when the CiCode function that created it ends.

If you assign an object created with the CreateObject() function to a module or global scope variable, then that object will remain in existence until the variable either has another object assigned or is set to NullObject, *provided the CreateObject() call is not made within a loop.*

Objects created by calls to CreateObject() within WHILE or FOR loops are only released on termination of the CiCode function in which they are created, regardless of the scope of the variable to which the object is assigned. The use of CreateObject() within a loop may therefore result in the exhaustion of system resources, and is not generally recommended unless performed as shown in the examples below.

Syntax

CreateObject(*sClass*)

sClass:

The class of the object. You can use the object's human readable name, its program ID, or its GUID. If the class does not exist, the function will fail.

For example:

- * "Calendar Control 8.0" - human readable name
- * "MSCAL.Calendar.7" - Program ID
- * "{8E27C92B-1264-101C-8A2F-040224009C02}" - GUID

Return Value

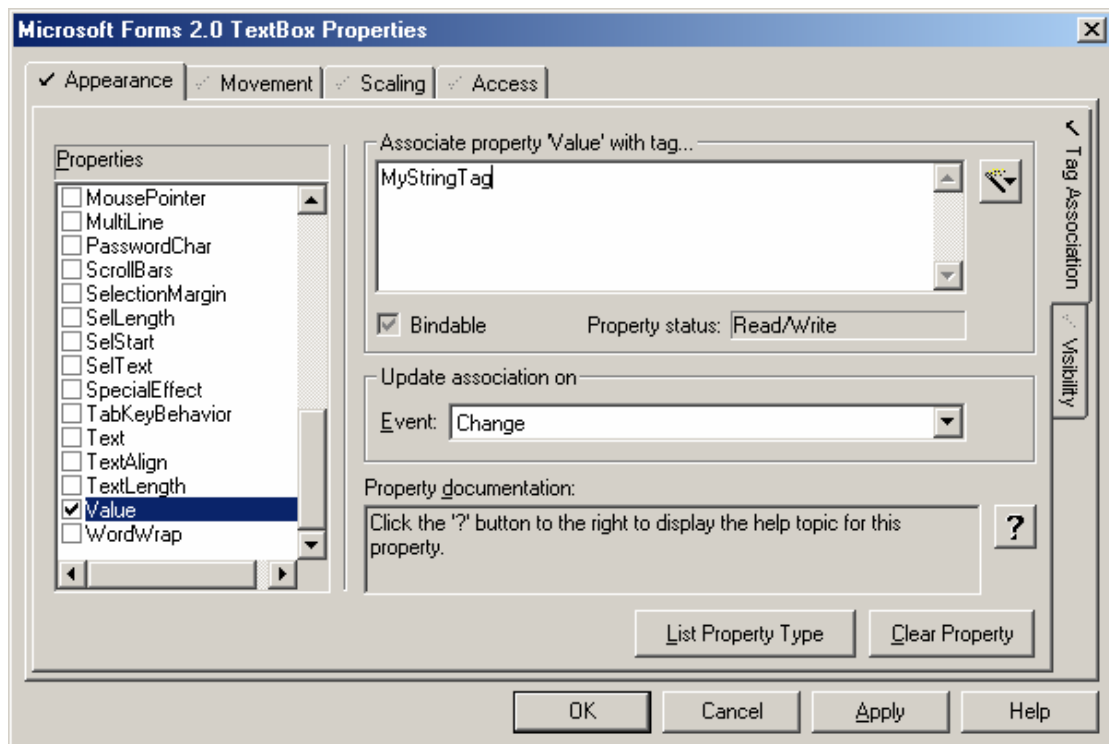
The newly created object, if successful, otherwise an [error](#) is generated.

Using ActiveX Objects in Citect SCADA

Once the instance of the ActiveX Control has been created, there are several ways of interacting with an ActiveX object via Citect SCADA.

All of these methods utilize the Properties, Methods and Events that we have

A practical example of using this would be linking the 'Value' Property of a MS Forms Textbox to a SCADA String Tag, to update on the 'Change' event, as shown below:



Whenever the text is changed within the Text Box, the contents of this Text Box is exposed to CitectSCADA via the 'Value' property. By linking this to a SCADA Tag, the SCADA Tag will now contain the contents of the TextBox, and will be updated whenever the Text changes.

Using ActiveX Controls via CiCode

Referencing an ActiveX Instance

In order to use CiCode to manipulate an ActiveX, we require a handle that we can use to point the CiCode commands to the correct instance of the ActiveX. This is when the 'Object Name' comes into play.

In order to return the Handle (of type OBJECT), from the 'Object Name', we can use the function:

```
OBJECT hActiveX;  
hActiveX = ObjectByName("AN201")
```

Where hActiveX is where we are going to store the handle, and "AN201" is the 'Object Name' as discussed previously.

We can abbreviate this by removing the hActiveX variable, and calling the ObjectByName function directly as an argument of another ActiveX function, however this is less efficient if you are doing this multiple times.

```
hActiveX = ObjectByName("AN201");  
sText = _ObjectGetProperty(hActiveX,"Value");
```

Can be abbreviated to:

```
sText = _ObjectGetProperty(ObjectByName("AN201"),"Value");
```

If we created the object via CiCode, we could use the Return Value of the CreateObject or CreateControlObject as the handle instead.

Now that we know how to refer to a specific instance of an ActiveX object, we can look into how to use this to manipulate the ActiveX object.

Using Properties from CiCode

There are two ways of manipulating Properties via CiCode, these are reading ('Get') and writing ('Set').

The following functions are used:

_ObjectGetProperty
_ObjectSetProperty

These are fairly simple, to get the current value of, say the 'Value' property of a MS Forms Textbox, we use the aforementioned example:

```
sText = _ObjectGetProperty(ObjectByName("AN201"),"Value");
```

This will return the value of the "Value" property of the TextBox at AN201, to the String Tag, sText.

Alternatively, in order to see if the TextBox has 'Multiline' enabled, then we can run the following CiCode:

```
IsMultiline = _ObjectGetProperty(ObjectByName("AN201"),"Multiline");
```

Which will return 0 or 1, for True or False.

In order to 'Set' a property, no return value is required (although it will return 0 for success, or an error code), however, this time we need to pass the Property Name, and the Value that we want it set to, i.e:

```
_ObjectSetProperty(ObjectByName("AN201"),"Value","This is my new text");
```

Or:

```
sText = "This is my new Text"  
hActiveX = ObjectByName("AN201")  
_ObjectSetProperty(hActiveX,"Value",sText);
```

Using Methods from CiCode

In order to call a Method from CiCode, we use the following function:

_ObjectCallMethod

In order to use this function, we need to point it at the correct instance of the ActiveX control using a handle, pass the name of the Method we want to call, as a string, parse arguments, and capture a return value, if required.

i.e In order to clear all items from a ComboBox, the 'Clear' method can be used as follows:

```
_ObjectCallMethod(ObjectByName("AN210"),"Clear");
```

Alternatively, to add an item to the ComboBox, you can use the 'AddItem' Method, and parse in the text that you wish to add, and an index:

```
_ObjectCallMethod(ObjectByName("AN210"),"AddItem","Tag1", Index);
```

To add another item, increment the iIndex value, and call a similar function again:

```
Index = Index + 1;  
_ObjectCallMethod(ObjectByName("AN210"),"AddItem","Tag2", Index);
```

Writing Event Handlers in CiCode:

An Event handler is usually used to run a function inside the Container Program (Citect SCADA), when a particular Event has triggered from within an ActiveX control.

In order to do this, we write essentially a normal CiCode function, but we have to pay careful attention to the name of the function, and the arguments, since the arguments will be parsed by the ActiveX control.

The function name must be of the format: *“EVENT_CLASS” + “_” + “Event Name”*

Note: There is a limitation in CitectSCADA where the name of the Event Handler must be no more than 33 characters long

Also of note, the first argument of your function must be of Type OBJECT, as the Handle of the calling ActiveX control is always provided by the ActiveX control when the event is triggered.

I.e in order to define a MouseDown event-handler on an ActiveX object at AN202 on the page ‘Page1’, we can define the function in the CiCode editor as follows:

```
FUNCTION page1_AN202_Mousedown(OBJECT th s, INT a, INT b,INT c,INT d)
    Message("H ", "The MouseDown event has triggered",0);
END
```

This will display a Message Box whenever the Mouse is clicked on the particular object.

Since we the handle of the calling ActiveX control has been parsed in as OBJECT this, we could add to our Event Handler:

```
FUNCTION page1_AN202_Mousedown(OBJECT th s, INT a, INT b,INT c,INT d)
    sText = _ObjectGetProperty(th s, "Value");
    Message("H ", "The MouseDown event has triggered and the value of 'Value' is " +
sText,0);
END
```

Advanced Concepts

Outlined below are some more advanced ActiveX concepts and examples. Process Analyst is an ActiveX Control that utilizes a number of these concepts and is a Control that we will likely want to manipulate, so it will be the example used to demonstrate these topics.

Accessing Properties with Indices

CitectSCADA cannot use `_ObjectGetProperty` to retrieve Properties with an index. However, some controls (such as Process Analyst) provide methods to access these using additional methods.

In order to access the Array Property 'Item', Process Analyst provides the method 'Get_Item' which can accept an argument, which is the index. For example, to access `Item(1)` we use:

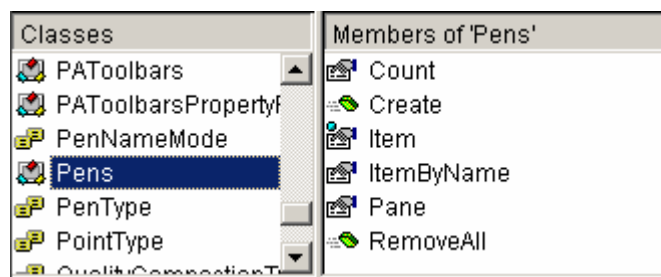
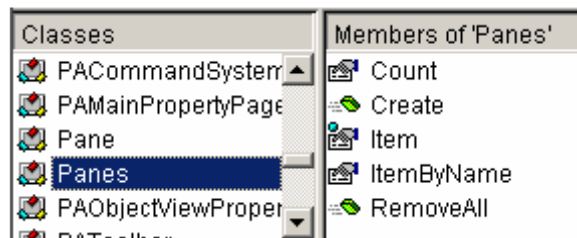
```
hPen = _ObjectCallMethod(hPens, "get_Item", 1);
```

This will make more sense after the next section.

Accessing members of 'Nested' Classes

Sometimes ActiveX controls with have multiple 'Classes', and the methods that you want to use may be 'nested' within one of these classes.

For example, Process Analyst has multiple classes, many with duplicate 'Members'



So we would need to distinguish between the 'Create' Method in Pens, and the 'Create' method in Panels. In VBA you could use `Panels.Create` and `Pens.Create`, but CiCode does not have this functionality, hence you need to do the following:

First, say our Process Analyst instance is at AN 215, we use the following code to return the Handle of that instance, as done previously:

```
hProcessAnalyst = ObjectByName("AN215");
```

Then, in order to return a handle to the 'Panels' class within the ActiveX, we use:

```
hPanels = _ObjectGetProperty(hProcessAnalyst,"Panels");
```

Now, we can use the methods of that class in the normal way, but using this new handle instead of the original handle of Process Analyst, however, since the Property we want to obtain (Item) is an array, we need to use the 'Get_Item' method, as explained in the previous section. We then end up with:

```
hPane = _ObjectCallMethod(hPanels, "get_Item", 1);
```

This now gives us a handle to the first Pane. From this, we can get a handle to the Pens class, within that Pane:

```
hPens = _ObjectGetProperty(hPane, "Pens");
```

And by using the aforementioned technique, we can now obtain a handle to a specific Pen within that specific Pane.

```
hPen = _ObjectCallMethod(hPens, "get_Item", 1);
```

Now we can simply call any of the iPen Class members on this Pen, in the usual way, i.e.:

```
_ObjectCallMethod(hPen, "Select");  
sTrendTagName = _ObjectCallMethod(hPen, "GetInformation", "Tag");  
sTrendTagComment = _ObjectCallMethod(hPen, "GetInformation", "Comment");  
sTrendVisible = _ObjectGetProperty(hPen, "Visible");  
_ObjectSetProperty(hPen, "Visible",0);  
etc...
```

iDispatch Interface and Citect Compatibility

As per KBQ4575, not all ActiveX's listed in the "Insert ActiveX Control" menu can be used in CitectSCADA.

In order to use an ActiveX object on a CitectSCADA page, CitectSCADA requires that the ActiveX control has an iDispatch interface.

If you try to use an ActiveX without this interface, which is not compatible, you will get the following error.



Linking Properties to SCADA Tags at Runtime via CiCode

As discussed in the section “Linking Properties to SCADA Tags in Design-Time” it is possible to link Properties to SCADA Tags at Design-Time. However, it is also possible to do this at Runtime, via the CiCode function:

ObjectAssociatePropertyWithTag

Our previous example that we configured via Design-Time:

“...A practical example of using this would be linking the ‘Value’ Property of a MS Forms Textbox to a SCADA String Tag, to update on the ‘Change’ event..”

This would be achieved via the following Cicode function (assuming AN201):

```
ObjectAssociatePropertyWithTag(ObjectByName(“AN201”,“Value”,“sTagName1”,“Change”);
```

Note: An association will fail if property change notification is not supported and the OnChangeEvent argument is left blank.

From Help:

Syntax:

```
ObjectAssociatePropertyWithTag(sObject, sPropertyName, sTagName [, sOnChangeEvent] )
```

sObject:

The object instance that associates a property with a tag.

sPropertyName:

The name of the ActiveX property to associate with the tag.

sTagName:

The name of the CitectSCADA variable tag to associate with the property.

sOnChangeEvent:

The name of the "on change" event that informs CitectSCADA of a change to the ActiveX object. This is required where the ActiveX object does not automatically generate a property change notification. Choose an event that happens to be fired whenever the ActiveX object property changes, for example, the MS Calendar Control fires an AfterUpdate event whenever a day button is pressed.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

ActiveX Cicode Function List

Here is a listing of all the ActiveX related Cicode functions. Some of the functions such as `ObjectIsValid` and `ObjectHasInterface` can be leveraged to add error handling and to make your Cicode more robust, especially when creating instances during Runtime, and where you cannot guarantee that the instance was created successfully.

<u>ObjectCallMethod</u>	Calls a specific method for an ActiveX object.
<u>ObjectGetProperty</u>	Retrieves a specific property of an ActiveX object.
<u>ObjectSetProperty</u>	Sets a specific property of an ActiveX object.
<u>AnByName</u>	Retrieves the animation point number of an ActiveX object.
<u>CreateControlObject</u>	Creates a new instance of an ActiveX object.
<u>CreateObject</u>	Creates the automation component of an ActiveX object.
<u>ObjectAssociateEvents</u>	Allows you to change the ActiveX object's event class.
<u>ObjectAssociatePropertyWithTag</u>	Establishes an association between a variable tag and an ActiveX object property.
<u>ObjectByName</u>	Retrieves an ActiveX object.
<u>ObjectHasInterface</u>	Queries the ActiveX component to determine if its specific interface is supported.
<u>ObjectIsValid</u>	Determines if the given handle for an object is valid.
<u>ObjectToStr</u>	Converts an object handle to a string.

Link / Recommended Reading

I recommend reading the advanced Process Analyst whitepapers, available in the KnowledgeBase:

WhitePapers:

- Adding a Column to Display the Current Trend Value
- Creating custom toolbar buttons in Process Analyst
- How to configure CiRecipe ActiveX control in CitectSCADA

KB Articles:

Q4634 - Tag association in ActiveX object only accepts name =<32 characters
Q4483 - My ActiveX's event wont trigger (EventName is being concatenated to 33 Characters in length)
Q4102 - Finding members of an ActiveX control library
Q4334 - How do I view Microsoft Forms ActiveX Controls without installing Microsoft Office?
Q4181 - Creating ActiveX controls that integrate with CitectSCADA
Q3900 - What Are the Differences between Making Tag Associations at Runtime and Design Time?
Q3876 - Common Questions about ActiveX Controls in CitectSCADA
Q3809 - How Can I Remove the Obsolete ActiveX Entries from the Registry?
Q3696 - _ObjectCallMethod May Cause CitectHMI/SCADA Crash.
Q3386 - Using the ActiveX Registration Tool REGSVR32.EXE
Q3237 - How to use IE Browser ActiveX control with Citect
Q2994 - Using Active X Cicode Functions