Working with ActiveX Controls

ActiveX controls are third-party custom controls that you can integrate in a Natural dialog.

This chapter covers the following topics:

- Terminology
- How To Define an ActiveX Control
- How To Create an ActiveX Control
- Accessing Simple Properties
- Colors
- Pictures
- Fonts
- Variants
- Arrays
- Using the PROCESS GUI Statement

Terminology

ActiveX controls and Natural use different terminology in two cases:

ActiveX Control	Natural
Property	Attribute
Method	`PROCESS GUI` Statement Action

How To Define an ActiveX Control

The handle of an ActiveX control is defined similar as a built-in dialog element, but its individual aspects are coded in double angle brackets.

Example:

```
01    #OCX-1 HANDLE OF <<OCX-Table.TableCtrl.1 [Table Control]>>
```

In the above example, `Table.TableCtrl.1` is the program ID (ProgID) under which the ActiveX control is registered in the system registry. The prefix `OCX-` identifies the control as an ActiveX control. `[Table Control]` is an optional part of the definition and provides a readable name.

How To Create an ActiveX Control

You create an instance of an ActiveX control by using the PROCESS GUI statement action ADD. To do so, the value of the TYPE attribute must be the ActiveX control's ProgID prefixed with the string OCX- and put in double angle brackets. The ProgID is the name under which the control is registered in the system registry. You can additionally provide a readable name in square brackets. In addition to that, you can set Natural attributes such as RECTANGLE-X as well as the ActiveX control's properties. The property name must be preceded by the string PROPERTY- and this combination must be put in double angle brackets. Furthermore, you can suppress the ActiveX control's events. To do this, the event name must be preceded by the string SUPPRESS-EVENT this combination must be delimited by double angle brackets. The value of the SUPPRESS-EVENT property is either the Natural keyword SUPPRESSED or NOT-SUPPRESSED.

Example:

```
PROCESS GUI ACTION ADD
    WITH PARAMETERS
       HANDLE-VARIABLE = #OCX-1
       TYPE = <<OCX-Table.TableCtrl.1 [Table Control]>>
       PARENT = #DLG$WINDOW
       RECTANGLE-X = 44
       RECTANGLE-Y = 31
       RECTANGLE-W = 103
       RECTANGLE-H = 46
       <<PROPERTY-HeaderColor>> = H'FF0080'
       <<PROPERTY-Rows>> = 16
       <<PROPERTY-Columns>> = 4
       <<SUPPRESS-EVENT-RowMoved>> = SUPPRESSED
       <<SUPPRESS-EVENT-ColMoved>> = SUPPRESSED
    END-PARAMETERS
```

Accessing Simple Properties

Simple properties are properties that do not have parameters. Simple properties of an ActiveX control are addressed like attributes of built-in controls. The attribute name is built by prefixing the property name with PROPERTY- and enclosing it in angle brackets.The properties of an ActiveX control are displayed in the Component Browser. The following examples assume that the ActiveX control #OCX-1 has the simple properties CurrentRow and CurrentCol.

Example:

```
* Get the value of a property.
#MYROW := #OCX-1.<<PROPERTY-CurrentRow>>
* Put the value of a property.
#OCX-1.<<PROPERTY-CurrentCol>> := 17
```

The data types of ActiveX control properties are those defined by OLE Automation. In Natural, each of these data types is mapped to a corresponding Natural data type. The following table shows which OLE Automation data type is mapped to which Natural data type.

OLE Automation data type	NATURAL data type
VT_BOOL	L
VT_BSTR	A dynamic
VT_CY	P15.4
VT_DATE	T
VT_DECIMAL	Pn.m
VT_DISPATCH	HANDLE OF OBJECT
VT_ERROR	I4
VT_I1	I2
VT_I2	I2
VT_I4	I4
VT_INT	I4
VT_R4	F4
VT_R8	F8
VT_U1	B1
VT_U2	B2
VT_U4	B4
VT_UINT	B4
VT_UNKNOWN	HANDLE OF OBJECT
VT_VARIANT	(any Natural data type)
OLE_COLOR (VT_UI4)	B3
VT_FONT (VT_DISPATCH IFontDisp*)	HANDLE OF FONT HANDLE OF OBJECT (IFontDisp*) A dynamic
VT_PICTURE (VT_DISPATCH IPictureDisp*)	HANDLE OF OBJECT (IPictureDisp*) A dynamic

Read the table in the following way: Assume an ActiveX control `#OCX-1` has a property named "Size", which is of type VT_R8. Then the expression `#OCX-1.<<PROPERTY-SIZE>>` has the type F8 in Natural.

Note:
The Component Browser displays the corresponding Natural data types directly.

Some special data types are considered individually in the following:

Colors

A property of type Color appears in Natural as a B3 value. The B3 value is interpreted as an RGB color value. The three bytes contain the red, green and blue elements of the color, respectively. Thus for example H'FF0000' corresponds to red, H'00FF00' corresponds to green, H'0000FF' corresponds to blue and so on.

Example:

```
...
 01 #COLOR-RED (B3)
...
 #COLOR-RED := H'FF0000'
 #OCX-1.<<PROPERTY-BackColor>> := #COLOR-RED
...
```

Pictures

A property of type Picture appears in Natural as `HANDLE OF OBJECT`. Alternatively you can assign an Alpha value to a Picture property. The Alpha value must then contain the file name of a Bitmap (*.bmp*) file.

Example (usage of Picture properties):

```
...
 01 #MYPICTURE HANDLE OF OBJECT
...
 * Assign a Bitmap file name to a Picture property.
 #OCX-1.<<PROPERTY-Picture>>:= '11100102.bmp'
 *
 * Get it back as an object handle.
 #MYPICTURE := #OCX-1.<<PROPERTY-Picture>>
 *
 * Assign the object handle to a Picture property of another control.
 #OCX-2.<<PROPERTY-Picture>>:= #MYPICTURE
...
```

Fonts

A property of type Font appears in Natural as `HANDLE OF OBJECT`. You can alternatively assign a `HANDLE OF FONT` to a Font property. Additionally you can assign an Alpha value to a Font property. The Alpha value must then contain a font specification in the form that is returned by the `STRING` attribute of a `HANDLE OF FONT`.

Example 1 (using `HANDLE OF OBJECT`):

```
...
01 #MYFONT HANDLE OF OBJECT
...
* Create a Font object.
CREATE OBJECT #MYFONT OF CLASS 'StdFont'
#MYFONT.Name := 'Wingdings'
#MYFONT.Size := 20
#MYFONT.Bold := TRUE
*
* Assign the Font object as value to a Font property.
#OCX-1.<<PROPERTY-TitleFont>> := #MYFONT
...
```

Example 2 (using `HANDLE OF FONT`):

```
...
01 #FONT-TAHOMA-BOLD-2 HANDLE OF FONT
...
* Create a Font handle.
PROCESS GUI ACTION ADD WITH PARAMETERS
    HANDLE-VARIABLE = #FONT-TAHOMA-BOLD-2
    TYPE = FONT
    PARENT = #DLG$WINDOW
    STRING = '/Tahoma/Bold/0 x -27/ANSI VARIABLE SWISS DRAFT/W/2/3/'
END-PARAMETERS GIVING *ERROR
...
* Assign the Font handle as value to a Font property.
#OCX-1.<<PROPERTY-TitleFont>> := #FONT-TAHOMA-BOLD-2
...
```

Example 3 (using a font specification string):

```
...
01 #FONT-TAHOMA-BOLD-2 HANDLE OF FONT
...
* Create a Font handle.
PROCESS GUI ACTION ADD WITH  PARAMETERS
    HANDLE-VARIABLE = #FONT-TAHOMA-BOLD-2
    TYPE = FONT
    PARENT = #DLG$WINDOW
    STRING = '/Tahoma/Bold/0 x -27/ANSI VARIABLE SWISS DRAFT/W/2/3/'
END-PARAMETERS GIVING *ERROR
...
* Assign the font specification as value to a Font property.
#OCX-1.<<PROPERTY-TitleFont>> := #FONT-TAHOMA-BOLD-2.STRING
...
```

Variants

A property of type Variant is compatible with any Natural data type. This means that the type of the expression `#OCX-1.<<PROPERTY-Value>>` is not checked by the compiler, if "Value" is a property of type Variant. So the assignments `#OCX-1.<<PROPERTY-Value >> := #MYVAL` and `#MYVAL := #OCX-1.<<PROPERTY-Value >>` are allowed independently of the type of the variable `#MYVAL`. It is however up to the ActiveX control to accept or reject a particular property value at runtime, or to deliver the value in the requested format. If it does not, the ActiveX control will usually raise an exception. This exception is returned as a Natural error code to the Natural program. Here it can be handled in the usual way in an `ON ERROR` block. You should check the documentation of the ActiveX control to find out which data formats are actually allowed for a particular property of type Variant.

An expression like `#OCX-1.<<PROPERTY-Value>>` (where "Value" is a Variant property) can occur as source operand in any statement. However, it can be used as target operand only in assignment statements.

Examples (usage of Variant properties):

(Assume that "Value" is a property of type Variant of the ActiveX control `#OCX-1`)

```
...
01 #STR1 (A100)
01 #STR2 (A100)
...
* These statements are allowed, because the Variant property is used
* as source operand (its value is read).
#STR1 := #OCX-1.<<PROPERTY-Value>>
COMPRESS #OCX-1.<<PROPERTY-Value>> 'XYZ' to #STR2
...
* This leads to an error at compiletime, because the Variant
* property is used as target operand (its value is modified) in
* a statement other than an assignment.
COMPRESS #STR1 "XYZ" to #OCX-1.<<PROPERTY-Value>>
...
* This statement is allowed, because the Variant property is used
* as target operand in an assignment.
COMPRESS #STR1 'XYZ' to #STR2
#OCX-1.<<PROPERTY-Value>> := #STR2
...
```

Arrays

A property of type SAFEARRAY of up to three dimensions appears in a Natural program as an array with the same dimension count, occurrence count per dimension and the corresponding format. (Properties of type SAFEARRAY with more than three dimensions cannot be used in Natural programs.) The dimension and occurrence count of an array property is not determined at compiletime but only at runtime. This is because this information is variable and is not defined at compiletime. The format however is checked at compiletime.

Array properties are always accessed as a whole. So no index notation is necessary and allowed with an array property.

Examples (usage of Array properties):

(Assume that "Values" is a property of the ActiveX control `#OCX-1` an has the type SAFEARRAY of VT_I4)

```
...
01 #VAL-L (L/1:10)
01 #VAL-I (I4/1:10)
...
* This statement is allowed, because the format of the property
* is data transfer compatible with the format of the receiving array.
* However, if it turns out at runtime that the dimension count or
* occurrence count per dimension do not match, a runtime error will
* occur.
VAL-I(*) := #OCX-1.<<PROPERTY-Values>>
...
* This statement leads to an error at compiletime, because
```

```
* the format of the property is not data transfer compatible with
* the format of the receiving array.
VAL-L(*) := #OCX-1.<<PROPERTY-Values>>
...
```

Using the PROCESS GUI Statement

The methods of ActiveX controls are called as actions in a `PROCESS GUI` statement. The same is the case with the complex properties of ActiveX controls (i. e. properties that have parameters). The methods and properties of an ActiveX control are displayed in the Component Browser.

This section covers the following topics:

- Performing Methods
- Getting Property Values
- Putting Property Values
- Optional Parameters
- Error Handling
- Using Events With Parameters
- Suppressing Events At Runtime

Performing Methods

To perform a method of an ActiveX control the `PROCESS GUI` statement is used. The name of the corresponding `PROCESS GUI` action is built by prefixing the method name with `METHOD-` and enclosing it in angle brackets. The ActiveX control handle and the method parameters (if any) are passed in the `WITH` clause of the `PROCESS GUI` statement. The return value of the method (if any) is received in the variable specified in the `USING` clause of the `PROCESS GUI` statement.

This means: To perform a method, you enter a statement

`PROCESS GUI ACTION <<METHOD-`*methodname***`>> WITH`***handlename* [*parameter*]...
[**`USING`** *method-return-operand*]..

Examples:

```
* Performing a method without parameters:
PROCESS GUI ACTION <<METHOD-AboutBox>> WITH #OCX-1
* Performing a method with parameters:
PROCESS GUI ACTION <<METHOD-CreateItem>> WITH #OCX-1 #ROW #COL #TEXT
* Performing a method with parameters and a return value:
PROCESS GUI ACTION <<METHOD-RemoveItem>> WITH #OCX-1 #ROW #COL USING #RETURN
```

Formats and length of the method parameters and the return value are checked at compiletime against the definition of the method, as it is displayed in the Component Browser.

Getting Property Values

To get the value of a property that has parameters, the name of the corresponding `PROCESS GUI` action is built by prefixing the property name with `GET-PROPERTY-` and enclosing it in angle brackets. The ActiveX control handle and the property parameters (if any) are passed in the `WITH` clause of the `PROCESS GUI` statement. The property value is received in the `USING` clause of the `PROCESS GUI` statement.

This means: To get the value of a property that has parameters, you enter a statement

PROCESS GUI ACTION <<GET-PROPERTY-*propertyname***>> WITH***handlename* [*parameter*]

... **USING** *get-property-operand*

Example:

```
PROCESS GUI ACTION <<GET-PROPERTY-ItemHeight>> WITH #OCX-1 #ROW #COL USING #ITEMHEIGHT
```

Formats and length of the property parameters and the property value are checked at compiletime against the definition of the method, as it is displayed in the Component Browser.

Putting Property Values

To put the value of a property that has parameters, the name of the corresponding `PROCESS GUI` action is built by prefixing the property name with `PUT-PROPERTY-` and enclosing it in angle brackets. The ActiveX control handle and the property parameters (if any) are passed in the `WITH` clause of the `PROCESS GUI` statement. The property value is passed in the `USING` clause of the `PROCESS GUI` statement.

This means: To put the value of a property that has parameters, you enter a statement

PROCESS GUI ACTION <<PUT-PROPERTY-*propertyname***>> WITH***handlename* [*parameter*]

... **USING** *put-property-operand*

Example:

```
PROCESS GUI ACTION <<PUT-PROPERTY-ItemHeight>> WITH #OCX-1 #ROW #COL USING #ITEMHEIGHT
```

Formats and length of the property parameters and the property value are checked at compiletime against the definition of the method, as it is displayed in the Component Browser.

Optional Parameters

Methods of ActiveX controls can have optional parameters. This is also true for parameterized properties. Optional parameters need not to be specified when the method is called. To omit an optional parameter, use the placeholder 1X in the `PROCESS GUI` statement. To omit *n* optional parameters, use the placeholder *n*X.

In the following example it is assumed that the method `SetAddress` of the ActiveX control `#OCX-1` has the parameters `FirstName`, `MiddleInitial`, `LastName`, `Street` and `City`, where `MiddleInitial`, `Street` and `City` are optional. Then the following statements are correct:

```
* Specifying all parameters.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName MiddleInitial LastName Street City
* Omitting one optional parameter.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName 1X LastName Street City
* Omitting the optional parameters at end explicitly.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName MiddleInitial LastName 2X
* Omitting the optional parameters at end implicitly.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName MiddleInitial LastName
```

Omitting a non-optional (mandatory) parameter results in a syntax error.

Error Handling

The `GIVING` clause of the `PROCESS GUI` statement can be used as usual to handle error conditions. The error code can either be caught in a user variable and then be handled, or the normal Natural error handling can be triggered and the error condition be handled in an `ON ERROR` block.

Example:

```
DEFINE DATA LOCAL
1 #RESULT-CODE (N7)
...
END-DEFINE
...
* Catching the error code in a user variable:
PROCESS GUI ACTION <<METHOD-RemoveItem>> WITH #OCX-1 #ROW #COL USING #RETURN GIVING #RESULT-CODE
*
* Triggering the Natural error handling:
PROCESS GUI ACTION <<METHOD-RemoveItem>> WITH #OCX-1 #ROW #COL USING #RETURN GIVING *ERROR-NR
...
```

Special error conditions that can occur during the execution of ActiveX control methods are:

- A method parameter, method return value or property value could not be converted to the data format expected by the ActiveX control. (These format checks are normally already done at compiletime. In these cases no runtime error can be expected. However, note that method parameters, method return values or property values defined as Variant are not checked at compiletime. This applies also for arrays and for those data types that can be mapped to several possible Natural data types.)
- A COM or Automation error occurs while locating and executing a method.
- The ActiveX control raises an exception during the execution of a method.

In these cases the error message contains further information provided by the ActiveX control, which can be used to determine the reason of the error with the help of the documentation of the ActiveX control.

Using Events With Parameters

Events sent by ActiveX controls can have parameters. In the controls event-handler sections, these parameters can be queried. Parameters passed by reference can also be modified. The events of an ActiveX control, the names and data types of the parameters and the fact if a parameter is passed by value or by reference is all displayed in the Component Browser.

Event parameters of an ActiveX control are addressed like attributes of built-in controls. The attribute name is built by prefixing the parameter name with `PARAMETER-` and enclosing it in angle brackets. Alternatively, parameters can be addressed by position. This means the attribute name is built by prefixing the number of the parameter with `PARAMETER-` and enclosing it in angle brackets. The first parameter of an event has the number 1, the second the number 2 and so on. These attribute names are only valid inside the event handler of that particular event.

In the following examples it is assumed that a particular event of the ActiveX control `#OCX-1` has the parameters `KeyCode` and `Cancel`. Then the event handler of that event might contain the following statements:

```
* Querying a parameter by name:
#PRESSEDKEY := #OCX-1.<<PARAMETER-KeyCode>>
* Querying a parameter by position:
#PRESSEDKEY := #OCX-1.<<PARAMETER-1>>
```

Parameters that are passed by reference can be modified in the event handler. In the following example it is assumed that the `Cancel` parameter is passed by reference and is thus modifiable. Then the event handler might contain the following statements:

```
* Modifying a parameter by name:
#OCX-1.<<PARAMETER-Cancel>>:= TRUE
* Modifying a parameter by position:
#OCX-1.<<PARAMETER-2>>:= TRUE
```

Suppressing Events At Runtime

To suppress or unsuppress an event of an ActiveX control at runtime, modify the corresponding suppress event attribute of the control. The name of the suppress event attribute is built by prefixing the event name with `SUPPRESS-EVENT-` and enclosing it in angle brackets. The events of an ActiveX control are displayed in the Component Browser.

The following example assumes that the ActiveX control `#OCX-1` has the event `ColMoved`.

```
* Suppress the event.
#OCX-1.<<SUPPRESS-EVENT-ColMoved>> := SUPPRESSED
* Unsuppress the event.
#OCX-1.<<SUPPRESS-EVENT-ColMoved>> := NOT-SUPPRESSED
```