Defining and Using Context Menus

This chapter covers the following topics:

Introduction

As from Natural v4.1.1, it is possible to create context menus for use within Natural applications. The context menus can be completely static (i.e., the menu contents are known in advance and can be built via the dialog editor) or wholly or partially dynamic (i.e., the menu contents and/or state depend on the runtime context and are not completely known at design time).

Construction

A context menu is very similar in concept to a submenu. Therefore, the same menu editor is used for editing a context menu as is used for editing a dialog’s menu bar. Menu items can be added to context menus, and events associated with them, in exactly the same way as for menu-bar submenus. There are no functional differences to the menu bar editor, except that the **OLE** combo box (which is applicable only to top-level menu-bar submenus) will always be disabled. It should be noted, however, that any accelerators defined for context menu items will be globally available as long as that menu item exists. Furthermore, the accelerator will trigger the menu item for which it is defined even if the context menu is not being displayed or if the focus is on a control using a different context menu or no context menu at all.

The context menu editor may be invoked via either a new menu item, **Context menus...** on the **Dialog** menu, or via its associated accelerator (CTRL+ALT+X by default), or toolbar icon. However, because the context menu editor can only edit one context menu editor at a time, the context-menu editor is not invoked directly. Instead, the Dialog Context Menus window is shown, where operations on the context menu as a whole are made, and from which the menu editor for a given (selected) context menu can be invoked.

Internally, in order to distinguish between submenus and context menus, context menus have a new type, `CONTEXT MENU`. Otherwise, the generated code in both cases is identical. Here is some sample code illustrating the statements used to build up a simple context menu containing two menu items:

```
/* CREATE CONTEXT MENU ITSELF:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #CONTEXT-MENU-1
  TYPE = CONTEXTMENU
  PARENT = #DLG$WINDOW
END-PARAMETERS GIVING *ERROR
```

```
/* ADD FIRST MENU ITEM:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #MITEM-1
  TYPE = MENUITEM
  DIL-TEXT = ’Invokes the first item’
  PARENT = #CONTEXT-MENU-1
  STRING = ’Item 1’
END-PARAMETERS GIVING *ERROR
/* ADD SECOND MENU ITEM:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #MITEM-2
  TYPE = MENUITEM
  DIL-TEXT = ’Invokes the second item’
  PARENT = #CONTEXT-MENU-1
  STRING = ’Item 2’
END-PARAMETERS GIVING *ERROR
```

Note that if context menus or context-menu items are created dynamically in user-written code, the context menu or menu items will not be visible to the dialog editor. For example, the dynamically created menu item will not be visible in the context menu list box, and the dynamically created menu items will not be visible in the context menu editor.

Association

After creating a context menu, the context menu needs to be associated with a Natural object. Context menus are supported for almost all controls types capable of receiving the keyboard focus and for the dialog window itself. The full list includes ActiveX controls, bitmaps, canvasses, edit areas and input fields, list boxes, push buttons, radio buttons, scroll bars, selection boxes, table controls, toggle buttons, standard and MDI child windows, and MDI frame windows.

For all object types supporting context menus, the corresponding attribute dialogs in the dialog editor include a read-only combo box listing all context menus created by the dialog editor, together with an empty entry. The selection of the empty entry implies that no context menu is to be used for this object, and is the default.

Internally, the association is achieved by a new attribute, `CONTEXT MENU`, which should be set to the handle of a context menu. This attribute can be assigned at or after object creation time, and defaults to `NULL-HANDLE` if not specified, indicating the absence of a context menu. For context menus created by the dialog editor, the context menu is specified at control creation time as illustrated below:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
  HANDLE-VARIABLE = #LB-1
  TYPE = LISTBOX
  RECTANGLE-X = 585
  RECTANGLE-Y = 293
  RECTANGLE-W = 142
  RECTANGLE-H = 209
  MULTI-SELECTION = TRUE
  SORTED = FALSE
  PARENT = #DLG$WINDOW
  CONTEXT-MENU = #CONTEXT-MENU-1
  SUPPRESS-FILL-EVENT = SUPPRESSED
END-PARAMETERS GIVING *ERROR
```

The same syntax can also be used for controls created in user-written event code. In other cases, where the control was created by the dialog editor but the context menu was not, the context menu attribute must be assigned to the control after its creation, e.g., in the dialog's `AFTER-OPEN` event:

```
/* CONTEXT MENU SPECIFIED AFTER CREATION:

#LB-2.CONTEXT-MENU := #CONTEXT-MENU-2
```

Note that a context menu is not destroyed when an object using it is destroyed. Instead, it is destroyed when its parent object (typically, the dialog for which the context menu was defined) is destroyed. Similarly, the assignment of a new menu handle to the `CONTEXT MENU` attribute where one is already assigned does not result in the previous context menu being destroyed. Thus, using the above examples, neither of the following statements results in `CONTEXT-MENU-1` being destroyed:

```
PROCESS GUI ACTION DELETE WITH #LB-1            /* #CONTEXT-MENU-1 LIVES ON

#LB-1.CONTEXT-MENU := #CONTEXT-MENU-2           /* SAME HERE
```

Invocation

The context menu invocation process in Natural is as follows:

1. If the context menu is accessed via the mouse (i.e., secondary mouse button click), the target control is initially assumed to be the control immediately under the mouse cursor. Otherwise, if the context menu is accessed via the keyboard (i.e., either via the context menu key, if any, or via the key combination Shift+F10), the target control is initially assumed to be the control that currently has the keyboard focus.

2. The control's click position is set, relative to the target control's client area. If the context menu is accessed via the keyboard, the click position is set to (0, 0).

3. A `CONTEXT-MENU` event is raised for the target control, if not suppressed via the `SUPPRESS-CONTEXT-MENU-EVENT` attribute.

4. The target control's `CONTEXT-MENU` attribute is queried. Depending on its value and the type of the target control, the following action is taken:

   - If the attribute is set to `NULL-HANDLE` and the target control is a dialog, the context menu invocation process is aborted, without any context menu having been displayed.

   - If the attribute is set to `NULL-HANDLE` and the target control is a dialog element, the target control is assumed to be the dialog element's `PARENT`, and the context menu invocation process repeats starting with step 2 above.

   - If the attribute is set to the handle of a context menu, this context menu is taken as being the context menu that needs to be displayed (i.e., the *target* context menu), and processing continues with step 5 below.

5. A `BEFORE-OPEN` event is raised for the target context menu, if not suppressed.

6. The target context menu's `ENABLED` attribute is queried. If it is set to `FALSE`, the context menu is not displayed.

7. Otherwise, a `COMMAND-STATUS` event is raised for the target dialog, if not suppressed. The target dialog is the dialog containing the target control, if it is a dialog element, or the target control itself, if it is a dialog.

8. The context menu is displayed at the click position set in step 2 above.

The actual navigation within the context menu and the triggering of the events associated with the menu items is done by Windows and Natural with no intervention from the application.

Note that the above process continues up through the control hierarchy, starting with the initial target control, until if finds a dialog or dialog element with a context menu (if any), and then uses that context menu.

The purpose of the `CONTEXT-MENU` event is to allow application to select the appropriate context menu (by modifying the target control's `CONTEXT-MENU` attribute) from a number of possible candidates according to the context. For an example of using multiple context menus, see *Working with List View Controls*.

Similarly, the context menu's `BEFORE-OPEN` event gives the application the chance to modify the context menu according to the current program state. For example, menu items could be added or deleted, or particular menu items grayed or checked. Here is some sample code for the `BEFORE-OPEN` event:

```
/* DELETE FIRST MENU ITEM:
PROCESS GUI ACTION DELETE WITH #MITEM-1
/* CHECK SECOND MENU ITEM:
#MITEM-2.CHECKED := CHECKED
/* DISABLE THIRD MENU ITEM:
#MITEM-3.ENABLED := FALSE
/* INSERT NEW MENU ITEM BEFORE #MITEM-3:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #MITEM-4
  TYPE = MENUITEM
  DIL-TEXT = 'Invokes the first item'
  PARENT = #CONTEXT-MENU-1
  STRING = 'Item 3'
  SUCCESSOR = #MITEM-3
END-PARAMETERS GIVING *ERROR
```

For context menus not created by the dialog editor, the handling of the `BEFORE-OPEN` event must be done in the `DEFAULT` event for the dialog. Note also that if a control or dialog is disabled, no context menu is displayed, and the `BEFORE-OPEN` event is also not triggered. The same applies if the context menu itself is disabled. For example:

```
#CONTEXT-MENU-1.ENABLED := FALSE        /* DISABLE CONTEXT MENU DISPLAY
```

Note that it is possible to disable the context menu in this way during the `BEFORE-OPEN` event, allowing selective disabling of the context menu depending on the mouse cursor position within the control. For example, it might be desired to only display a context menu if the mouse cursor is over a selected list-box item. Determining whether this is the case is possible via the use of two `PROCESS GUI ACTION` calls:

- `INQ-CLICKPOSITION` has been extended to controls other than bitmaps and canvasses to return the (X, Y) position of the right mouse button click within the control. In addition, these parameters are now optional, and a new optional parameter has been introduced that is set to `TRUE` if the context menu was accessed via the mouse, or `FALSE` if it was accessed by the keyboard. In the latter case, the click position is set to (0, 0). All this information is updated immediately prior to the sending of the `BEFORE-OPEN` event.

- INQ-ITEM-BY-POSITION. This allows translation of the relative co-ordinate returned by INQ-CLICKPOSITION applied to a list box to the corresponding item.

As an example of the use of these two new actions, consider the situation where we want to detect whether the cursor was over a selected list-box item when the right mouse button was pressed in order to determine whether to display a context menu or not. This can be achieved by the following code in the BEFORE-OPEN event of the associated context menu:

```
PROCESS GUI ACTION INQ-CLICKPOSITION WITH
   #LB-1 #X-OFFSET #Y-OFFSET
PROCESS GUI ACTION INQ-ITEM-BY-POSITION WITH
   #LB-1 #X-OFFSET #Y-OFFSET #LBITEM
#MENU = *CONTROL
IF #LBITEM = NULL-HANDLE                  /* NO ITEM UNDER (MOUSE) CURSOR */
  #MENU.ENABLED := FALSE
ELSE
  IF #LBITEM.SELECTED = FALSE             /* ITEM UNDER CURSOR DESELECTED */
     #MENU.ENABLED := FALSE
  ELSE                                    /* ITEM UNDER CURSOR IS SELECTED */
     #MENU.ENABLED := TRUE
  END-IF
END-IF
```

In some cases, it may be desired to automatically select the item under the mouse cursor if it is not already selected, clearing any existing selection. For list boxes, it is possible to achieve this by using the new AUTOSELECT attribute, either directly or via the new **Autoselect** check box in the List Box Attributes window in the dialog editor. If this attribute is set to TRUE, Natural will automatically update the selection before sending the BEFORE-OPEN event, if the context menu was invoked over an unselected list-box item.

For table controls, any change in the selection must be done via the application itself in the BEFORE-OPEN event. To make this possible, another new PROCESS GUI ACTION has been introduced for table controls:

- TABLE-INQUIRE-CELL. This returns the cell’s row and column number (starting from 1) for a relative (X, Y) position within the table. This position can (and would typically be) the position returned by a previous call to PROCESS GUI ACTION INQ-CLICKPOSITION.

The COMMAND-STATUS event is an alternative location for the application to perform any updating such as graying and checking of commands (i.e., menu items, tool bar items and signals). If you are already using this event, you do not need to perform these actions in the BEFORE-OPEN event.

Manual Invocation

In addition to the automatic context menu invocation process described above, it is also possible to invoke a particular context menu manually at a specific position via the SHOW-CONTEXT-MENU action.

This is primarily intended for (but not restricted to) use with ActiveX controls where the automatic mechanism is not always applicable. This is because some ActiveX controls, depending on their internal implementation, do not raise the message used by Natural to trigger the context menu display. In such cases, if the ActiveX control raises an event when the secondary mouse button is pressed, the context menu can be manually displayed within the event handler for that event via this action.

For example, assuming we wish to display the context menu `#CTXMENU-1` for the Microsoft Rich Textbox ActiveX Control, `#OCX-1`, we could use the following code in the control's MouseDown event handler:

```
IF #OCX-1.<<PARAMETER-Button>> = 2
  #X := #OCX-1.<<PARAMETER-x>> + 2
  #Y := #OCX-1.<<PARAMETER-y>> + 2
  PROCESS GUI ACTION SHOW-CONTEXT-MENU WITH
    #CTXMENU-1 #OCX-1 #X #Y GIVING *ERROR
END-IF
```

where the following local data definitions are assumed:

```
01 #X (I4)
01 #Y (I4)
```

Note that the above code first checks whether the secondary mouse button was pressed, then invokes a context menu manually, based on the position passed by the control. The position is, however, first corrected slightly to account for the fact that the position supplied by the control is relative to the ActiveX control (which has a 2-pixel sunken border), whereas the position used to display the context menu is assumed to be relative to the ActiveX control's *container* window (which has no border).

Note that some ActiveX controls may return coordinates in units other than pixels, such as twips (twentieths of a point). The following example shows how to convert co-ordinates (`#X`, `#Y`) from twips to pixels:

```
#CONTROL := *CONTROL
/* Convert x-coordinate
MULTIPLY #X BY #CONTROL.DPI
DIVIDE #X BY 1440
/* Convert y-coordinate
MULTIPLY #Y BY #CONTROL.DPI
DIVIDE #Y BY 1440
```

where `#CONTROL` is defined as `HANDLE OF GUI`, and `#X` and `#Y` are assumed to be of format I4.

The value 1440 is the number of twips per logical inch, whereas the DPI attribute applied to a dialog element returns the number of pixels per logical inch.

Sharing of Context Menus

It is of course possible to associate the same context menu with more than one object (i.e., control or dialog). For example:

```
#LB-1.CONTEXT-MENU := #CTXMENU-1
#LB-2.CONTEXT-MENU := #CTXMENU-1
```

In such a scenario, we need to be able to determine for which control the context menu was invoked. We cannot use `*CONTROL` in the `BEFORE-OPEN` event, because this will contain the handle of the context menu itself. Instead, it is necessary to inquire which control has the focus, since Natural automatically places the focus on the control for which the context menu is being invoked. Here is some sample `BEFORE-OPEN` event code illustrating the use of this technique:

```
PROCESS GUI ACTION GET-FOCUS WITH #CONTROL
DECIDE ON FIRST VALUE OF #CONTROL
  VALUE #LB-1
     #MITEM-17.ENABLED := FALSE
  VALUE #LB-2
     #MITEM-17.CHECKED := CHECKED
  NONE
     IGNORE
END-DECIDE
```

However, a better approach, which works in all cases, is to query the context menu's `CONTROL` attribute instead:

```
#CONTROL := *CONTROL
DECIDE ON FIRST VALUE OF #CONTROL.CONTROL
  ...
END-DECIDE
```