How To Create and Delete Dialog Elements Dynamically

How To Create and Delete Dialog Elements Dynamically

This chapter covers the following topics:

- Introduction
- Global Attribute List
- Creating Dialog Elements Statically and Dynamically
- How to Handle Events of Dynamically Created Dialog Elements

---

Introduction

Dialog elements are usually added to a dialog by means of the dialog editor. However, they can also be created and deleted dynamically. This may be done, for example, when the layout of a dialog is strongly context-sensitive.

A dialog element is created dynamically with the `ADD` action of the `PROCESS GUI` statement. This action returns a handle to the newly created dialog element. As soon as the dialog element is created, this handle points to a set of attributes specified for the dialog element just created.

Note:
ActiveX controls are created in a slightly different way than the standard way described below. This is described in *Working with ActiveX Controls*.

For more information on the actions available, and on the parameters that can be passed, see *Executing Standardized Procedures*.

Global Attribute List

By modifying any handle attribute operand of the form *`handlename.attributename`* (for example, `#PB-1.STRING`), you change an attribute value of the specific dialog element. As long as the dialog element is not yet created and the handle variable has its initial value (`NULL-HANDLE`), the handle attribute operand *handlename.attributename* refers to the global attribute list.

The global attribute list is a collection of all attributes defined for any dialog element. Natural contains one such collection. Whenever a dialog element is created, it "inherits" its attributes from this global attribute list. It does not inherit them when you create the dialog element with the `PROCESS GUI` statement action `ADD` using the `WITH PARAMETERS` option.

Creating Dialog Elements Statically and Dynamically

To define a dialog element statically (in the dialog editor), with an individual set of attributes, you must first set the attributes in the global attribute list to the desired values and then create the dialog element. After creation, the values of the attributes in the global attribute list remain intact. The next created dialog element gets the same attributes from the global attribute list as the previous one, except those that have

been modified.

The status of the global attribute list as found in the "after open" event handler is influenced by the dialog elements defined statically. Therefore, before you start creating dialog elements dynamically in the "after open" event handler, you should reset the attributes by means of the PROCESS GUI action RESET-ATTRIBUTES to prevent your dialog elements from inheriting unexpected values from the global attribute list. If you want to avoid this inheritance problem, use the PROCESS GUI statement action ADD with the WITH PARAMETERS option.

Unexpected values may also result from having attribute values that mean different things if used by different types of dialog elements. For example, the value S of the attribute STYLE means "scaled" for the dialog element type bitmap control but "solid" for the dialog element type line control.

The PROCESS GUI action ADD is used to define a dialog element dynamically. This clause of the PROCESS GUI statement enables you to specify the attribute values within the statement. The inheritance of attributes from the global attribute list does not affect the PROCESS GUI statement action ADD. The attributes specified in the statement are transferred to the global attribute list before the action ADD is performed.

Note:
When you use the PROCESS GUI statement with Parameter Clause 2 of the ADD action, the global attribute list is not used or affected. For parameters which are needed to create the dialog element, but which were not specified in the WITH PARAMETERS section of the PROCESS GUI action ADD statement, the default value is taken. Apart from these, only the parameters which are passed explicitly in the parameter list are used to create the dialog element.

To create list-box and selection-box items dynamically, it may be more convenient to use the PROCESS GUI action ADD-ITEMS. This allows you to insert several items at a time.

Example:

```
/* #PB-A inherits the current settings of the global attribute list
#PB-A.STRING := 'TEST1'
PROCESS GUI ACTION ADD WITH #DLG$WINDOW PUSHBUTTON #PB-A
#PB-B.STRING := 'TEST2'
/* #PB-B has the same attributes as #PB-A except STRING. This leads to #PB-B
/* covering #PB-A.
PROCESS GUI ACTION ADD WITH #DLG$WINDOW PUSHBUTTON #PB-B
COMPUTE #PB-C.RECTANGLE-Y = #PB-B.RECTANGLE-Y + #PB-C.RECTANGLE-H + 20
/* #PB-B has the same attributes as #PB-A except RECTANGLE-Y
/* #PB-C will be located 20 pixels below #PB-B
PROCESS GUI ACTION ADD WITH #DLG$WINDOW PUSHBUTTON #PB-C
```

To delete dialog elements dynamically, you use the PROCESS GUI action DELETE. You can also use this technique to delete dialog elements created with the dialog editor (at design time). You should, however, avoid using the handle of the deleted dialog element because this is invalid.

Dialog elements often do not have to be created dynamically. In some cases, it is sufficient to make dialog elements VISIBLE = TRUE and VISIBLE = FALSE, depending on the context. This technique is more efficient and easier to handle. It also enables you to "insert" dialog elements anywhere in the navigation sequence.

Example:

```
DEFINE DATA LOCAL
   ...
   1 #PB-1 HANDLE OF PUSHBUTTON
   ...
END-DEFINE
...
#PB-1.VISIBLE := FALSE
...
IF...                          /* Logical condition
   #PB-1.VISIBLE := TRUE
END-IF
```

How to Handle Events of Dynamically Created Dialog Elements

When a dialog element is created dynamically, you cannot use the dialog editor to associate events to it. Instead, you must handle all events of all dynamically created dialog elements in the DEFAULT event. In this event, you must filter out which event occurred for which dialog element. The code for this is similar to the code generated by the dialog editor. The general structure is:

Example:

```
DECIDE ON FIRST *CONTROL
VALUE #PB-A
   DECIDE ON FIRST *EVENT
      VALUE 'CLICK'
         /* Click event-handler code
      NONE
         IGNORE
   END-DECIDE
VALUE #PB-B
   ...
VALUE #PB-C
   ...
END-DECIDE
```

In the case of event code for dynamically created ActiveX controls, *where event parameters are used*, it is necessary to precede the event code with an OPTIONS 2 statement containing the name of the event, otherwise the compiler will not be able to process parameter references (e.g., #OCX-1.<<PARAMETER-...>>) successfully. However, in contrast to the implicit generation of the OPTIONS statement by the dialog editor for events for statically created controls, no OPTIONS 3 statement should be coded in this case. Otherwise the dialog editor would falsely interpret the OPTIONS 3 statement as the end marker for the DEFAULT event, resulting in a scanning error on attempting to load the dialog.

Example:

```
DECIDE ON FIRST *CONTROL
VALUE #OCX-1  /* MS Calendar control
  DECIDE ON FIRST *EVENT
      VALUE '-602' /* DispID for KeyDown event
        OPTIONS 2 KeyDown
            /* KeyDown event-handler code containing parameter
            /* access (e.g. #OCX-1.<<parameter-shift>>)
        NONE
         IGNORE
 END-DECIDE
...
END-DECIDE
```