Dialog_Callback

Overview

The dbList class has a nice feature called Prompt_Callback that allows the programmer to set properties and call methods in the popup list from within the invoking object. This eliminates the need to establish odd schemes for passing information to the popup when activating it. For example, we can dynamically set the label of a selection list dialog when you activate it by using Prompt_Callback.

Wouldn't it be nice to have a similar methodology for the dbModalPanel class for when we don't use it as a selection list? It's certainly simple to do. In fact, we can do this for any "popup class" that is called from a focusable object.

The secret

Any modal object that has its popup state set to true (you send popup to the object) has a procedure popup in it that can be augmented. For example:

```
Object MyPopup is a ModalPanel
        Procedure Popup
                Forward Send Popup
        End_Procedure
End_Object
```

And from some other object we send popup to MyPopup:

```
Object MyParentObject is a Button
        Procedure OnClick
                Send Popup to MyPopup
        End_Procedure
End_Object
```

It turns out that in procedure Popup in MyPopup, prior to the forward send, the parent object MyParentObject still has the focus, and we can use this bit of knowledge to our advantage.

```
Object MyPopup is a ModalPanel

        // We add a new property
        Property Integer piInvokingObjectId    Public 0

        Procedure Popup
                // And we set the property to the ID of the invoking object
                Set piInvokingObjectId to (focus(Self))
                Forward Send Popup
        End_Procedure
End_Object
```

Now in the popup dialog we know the object id of the object that called the dialog. If we then set up a standard callback procedure in the invoking object, we will have almost everything we need. The one remaining requirement is how to call it, and that we do from within the popup. The completed process is:

```
Object MyPopup is a ModalPanel

        // We add a new property
        Property Integer piInvokingObjectId    Public 0

        // We augment procedure popup in the dialog
        // to grab the ID of the invoking object
        Procedure Popup
                Set piInvokingObjectId to (focus(Self))
                Forward Send Popup
        End_Procedure

        // We augment Entering_Scope, which fires
        // once when the ModalPanel activates
        Procedure Entering_Scope
                Forward Send Entering_Scope
                // Call the method back in the invoking object
                Send Dialog_Callback to (piInvokingObjectID(Self))
        End_Procedure

End_Object

Object MyParentObject is a Button
        Procedure OnClick
                Send Popup to MyPopup
        End_Procedure

        Procedure Dialog_Callback Integer iPopupObject
                // Send whatever we want to iPopupObject
        End_Procedure
End_Object
```

Doing it the right way

All of the above code will work, but it’s really not the correct way to do it since we’ve done it at the object level. The proper way is to subclass ModalPanel or dbModalPanel, and any other class that might be used for calling dialogs.

```
Class cMydbModalPanel is a dbModalPanel
        Procedure Construct_Object
                Forward send Construct_Object
                Property Integer piInvokingObjectId  Public 0
        End_Procdure

        Procedure Entering_Scope
                Forward Send Entering_Scope
                Send Dialog_Callback to (piInvokingObjectId(Self)) (Self)
        End_Procedure

        Procedure Popup
                Set piInvokingObjectId to (focus(Self))
                forward send Popup
        End_Procedure

End_Class

Class cMyButton is a Button
        Procedure Construct_Object
                Forward send Construct_Object
        End_Procdure

        // The presence of this “stub” prevents errors from occurring
        // when you don’t have a need for a callback procedure and
        // don’t put one in the button.
        Procedure Dialog_Callback Integer iCallingObject
        End_Procedure
End_Class
```

Repeat the code in cMyButton for any other class where you might call a dialog, dbForm, dbSpinForm, etc.

What can we do with this?

We can do virtually anything we want to the popup from within the invoking object. Since Entering_Scope occurs early on in the activation process, we can do any or all of the following from within procedure Dialog_Callback:

Set the label of the ModalPanel
Rebuild constraints in any DDs in the popup
Rebuild any lists – Send Beginning_Of_Data
Populate any properties contained in the ModalPanel
Etc. etc.

Caveat

All of this will be useless if you call your dialog object from a non-focusable object. The property (Focus(Self)) will end up containing the ID of some other object that doesn't contain procedure Dialog_Callback and you will see an error 98.

Accelerator Keys

If you have programmed the above and are calling the dialog that has a Dialog_Callback procedure from a button, and also do so with an On_Key process, there's one additional thing you need to do.

```
  On_Key Key_Ctrl+Key_C send keyaction to (My_Btn(Self))
```

Calling a button from an On_Key process does not transfer the focus to the button it activates, and you will get an error when the dialog attempts to call the Dialog_Callback procedure in the button. Do the following instead

```
Procedure Button_Keyaction
        Send Activate to (My_Btn(Self))
        send keyaction to (My_Btn(Self))
End_Procedure

On_Key Key_Ctrl+Key_C Send Button_Keyaction
```