Working with Dialog Bar Controls

This document covers the following topics:

Introduction

A dialog bar is similar to a tool bar control in that it can either docked to one of the interior sides of the dialog's frame or (optionally) floated in its own separate window. Unlike tool bar controls, however, dialog bar controls are conceived general-purpose container controls and are not dedicated to containing primarily tool bar items. Furthermore, there are a number of other visual and behavioral differences between tool bar controls and dialog bars, some of which are discussed below.

A good example of a dialog bar control is the library workspace window in Natural Studio.

Creating a Dialog Bar Control

Dialog bar controls are created in the dialog editor in the same way as other standard controls (such as list boxes or push buttons) are. That is, they are either created statically in the dialog editor via the **Insert** menu or by drag and drop from the Insert tool bar, or dynamically at run-time by using a `PROCESS GUI ACTION ADD` statement with the `TYPE` attribute set to `DIALOGBAR`.

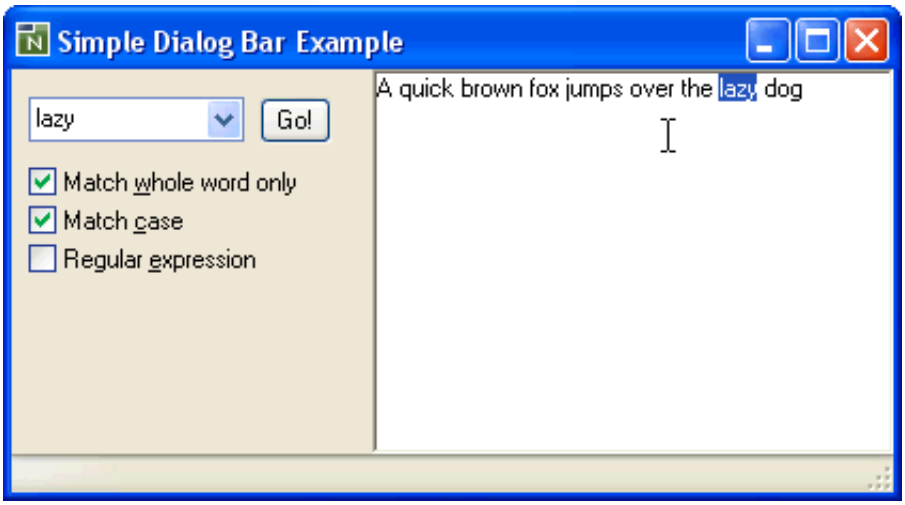
Types of Dialog Bar Control

A dialog bar control can exist in one of the following three basic forms (in order of complexity):

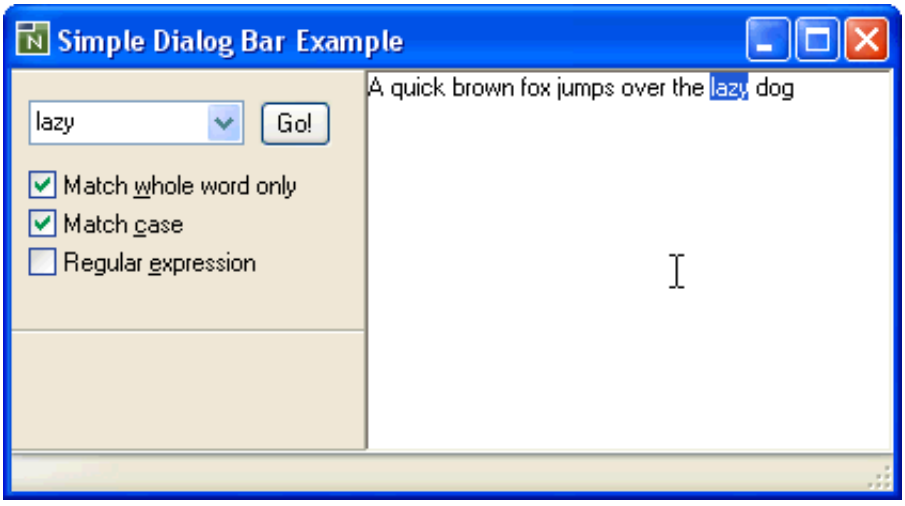
1. Neither dockable nor sizeable.
2. Dockable, but not sizeable.
3. Dockable and sizeable.

The dialog bar control is dockable if its `DRAGGABLE` attribute is set. It is sizeable if the "Dynamic (Y)" `STYLE` flag is set.

The following example shows an example of a non-dockable, non-sizeable dialog bar. The edit area on the right fills the entire client area of the dialog. The dialog bar cannot be dragged by the user and extends to fill the entire length of the side on which it is positioned:

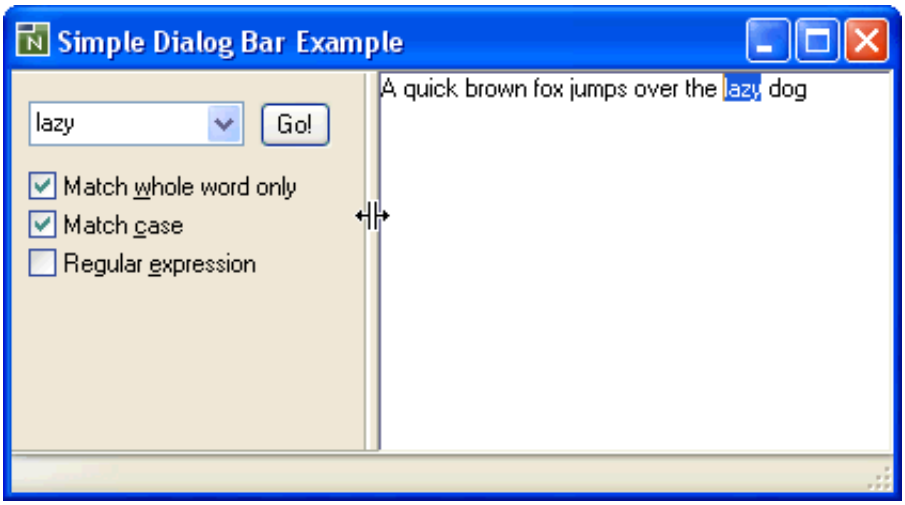


Setting the "dockable" state in the Dialog Bar Attributes window in the dialog editor causes the window to be draggable by the user. Note also that the dialog bar no longer extends to occupy the full length of the side to which it is docked (another dialog bar control or tool bar control could be docked underneath it):

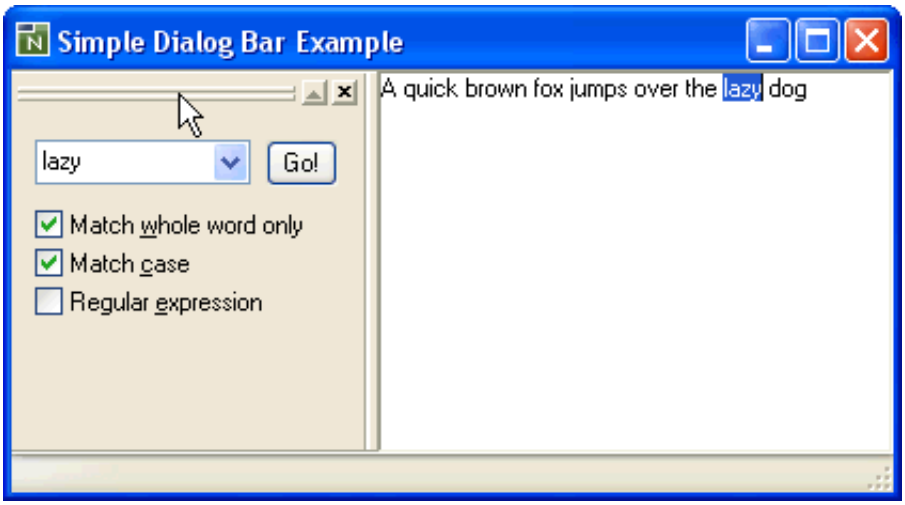


The user can drag the dialog bar control and re-dock it to another side of the owner dialog, or float it in its own separate window, as shown below:

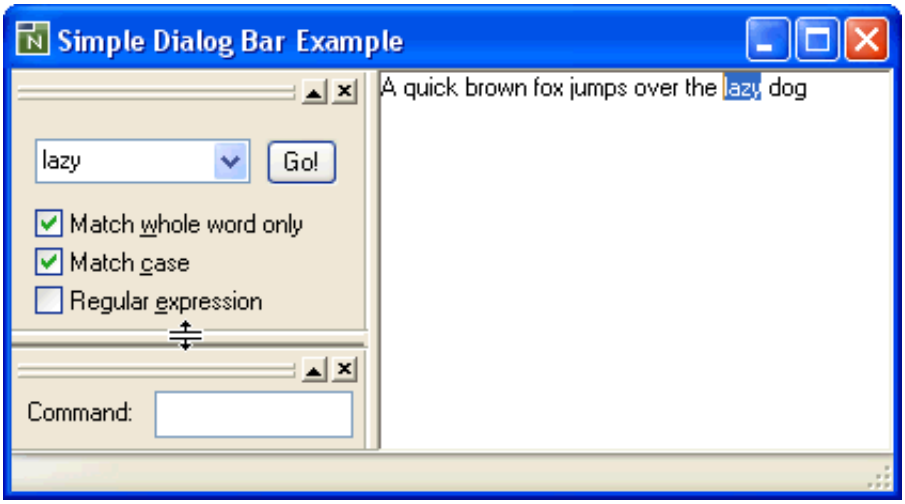
If the dialog bar control is made sizeable (by checking the "Dynamic (Y)" style flag in the Attributes window), a longitudinal splitter bar appears, allowing the dialog bar control to be resized. Note that sizeable dialog bar controls expand to fill the entire length of the side they are docked to that is not occupied by non-sizeable bars:



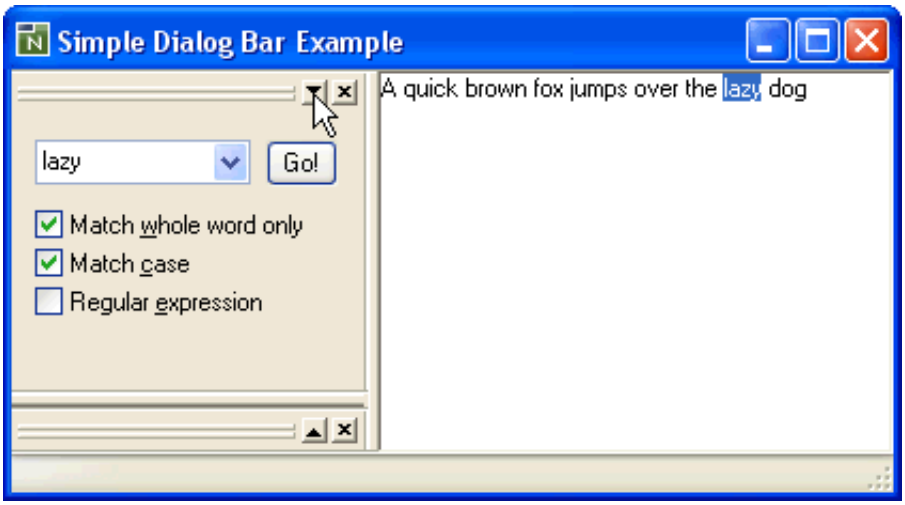
If a gripper bar, zoom and close button are added (by setting the "Gripper (g)", "Zoom button (z)" and "Close button (x)" style flags in the Attributes window), the dialog bar control takes on the familiar appearance of the control used to display the library workspace in Natural Studio. Note that the zoom button is disabled, because there is no other sizeable dialog bar control on the same row:



If a second sizeable dialog bar control is added, and docked alongside the first on the same row, a transverse splitter bar appears allowing the relative sizes of the two dialog bar controls to be changed. Note that the zoom button is now enabled:



Clicking on the zoom button toggles between the maximized and restored states of a sizeable dialog bar control. Maximizing a dialog bar control causes the other sizeable dialog bar controls on the same row to be minimized, and the released space to be taken up by the maximized bar, as shown below:



Note that the direction of the arrow is displayed by the zoom button on the maximized bar has changed direction in order to indicate that the next time this button is pressed, the bar will be restored, rather than maximized. When a bar is restored, all sizeable dialog bars on the row revert to their normalized sizes. These are usually the sizes prior to the maximize operation, unless there has been a change in the visible bars on the row in the meantime (e.g., visible bar hidden or hidden bar shown, new bar docked on row, etc.).

Note that if the length of a bar on a row is changed via a tranverse splitter bar, all visible bars on the row are automatically restored.

UI Transparency

A dockable dialog bar control may normally be dragged via either its gripper bar (if any) or its background. If the "UI transparent (T)" style is set, however, the bar may only be dragged via its gripper bar (if any). If such a (sizeable) bar does not have a gripper bar, resizing of the control is only possible via the splitter bar(s), which may be a desirable feature in some situations. Additionally, only allowing dragging via the gripper bar helps prevent unintentional initiation of drag operations.

Client-Size Event

The dialog’s client window is reduced in size to exclude the areas occupied by tool bar, status bar and dialog bar controls. If a dockable window (tool bar or dialog bar control) is floated, re-docked to another side of the owner window, or is shown or hidden, the size of the client window can change, even though the exterior dimensions of the window have not altered. Because the `SIZE` event is reserved for changes in a dialog’s exterior size, applications which need to keep track of the size of the client window should instead use the `CLIENT-SIZE` event for this purpose. The actual size of the dialog client window can then be determined within this event by means of the `INQ-INNER-RECT` action.

Close Button

The close button (if present) hides a dialog bar control rather than closing it, as is also the case for the close button on floated tool bar controls. It is up to the application to provide a method of re-showing the bar. The next section provides some code for doing this (amongst other things).

Sample Code

Below is a full listing of an external subroutine that can, in most cases, be used "as-is" in order to allow user control over the display of tool bars and dialog bars. The code is designed to be powerful enough to cope with MDI applications, but also works with non-MDI (i.e., SDI) applications.

The subroutine appends the tool and dialog bar captions (`STRING` attribute) to the dialog's context menu. If the dialog does not have a context menu, one is created and assigned to the dialog automatically. It assumes that, in an MDI application, there are some tool bars and dialog bars that are global (i.e., relevant for all types of MDI child dialogs) and some which are private (i.e., relevant only for one type of MDI child dialog). For example, in Natural Studio, the Object tool bar is an example of a global tool bar, whereas the dialog editor options tool bar is private to the dialog editor. When the user switches between MDI child dialogs, the context menu is changed to only show the global tool bars plus any private tool bars relevant to the currently active dialog. Furthermore, the same private bars are displayed as the last time this dialog was displayed (if the **Save layout** check box in the Dialog Attributes window is checked, the subset of bars shown is even retained between sessions).

The subroutine should be called in the `AFTER-ANY` event handler of the main dialog (i.e., the MDI frame dialog for MDI applications), as follows (assuming the main dialog's handle variable name is set to the default value of `#DLG$WINDOW`):

```
PERFORM PROCESS-BAR-COMMANDS #DLG$WINDOW
```

In addition, the following steps are optional:

1. The bars are listed in the context menu in the order in which they appear in the control sequence. Therefore, you may wish to re-sequence the tool and dialog bars (e.g., to ensure that the global tool bars are displayed before the private ones in MDI applications).

2. The code does not insert a separator before the list of available bars on the context menu. Therefore, if you are already using a context menu for the dialog, you would probably want to ensure that your context menu ends with a separator.

3. For MDI applications, for each private bar, you should enter the name of the dialog (e.g. "CHILD" if the dialog's file name is *CHILD.NS3*) to which the tool bar "belongs" into the **Control ID** field of the Attributes window for the bar in the dialog editor. For each global bar, leave this field empty. If you wish the bar to be displayed only when no MDI child dialog is active, enter the name of the MDI frame dialog here.

4. For MDI applications, you should uncheck the **Enabled** check box in the Attributes window for each bar that should not be displayed by default.

```
DEFINE DATA
PARAMETER
  1 #DIALOG     HANDLE OF GUI
LOCAL
  1 #CONTROL       HANDLE OF GUI
  1 #ACTIVE-DLG    HANDLE OF GUI
  1 #CTXMENU       HANDLE OF CONTEXTMENU
  1 #MITEM-DYN     HANDLE OF MENUITEM
LOCAL USING NGULKEY1
END-DEFINE
*
DEFINE SUBROUTINE PROCESS-BAR-COMMANDS
  DECIDE ON FIRST *EVENT
```

```
    VALUE 'COMMAND-STATUS'
       PERFORM COMMAND-STATUS
    VALUE 'IDLE'
       PERFORM IDLE
    VALUE 'CLICK'
      PERFORM CLICK
    VALUE 'BEFORE-OPEN'
      PERFORM BEFORE-OPEN
    VALUE 'AFTER-OPEN'
      PERFORM AFTER-OPEN
   NONE
      IGNORE
   END-DECIDE
*
   DEFINE SUBROUTINE COMMAND-STATUS
     /* Must enable our commands, otherwise they're automatically disabled!
     #CTXMENU := #DIALOG.CONTEXT-MENU
     #MITEM-DYN := #CTXMENU.FIRST-CHILD
     REPEAT WHILE #MITEM-DYN <> NULL-HANDLE
        IF #MITEM-DYN.CLIENT-HANDLE <> NULL-HANDLE
          #MITEM-DYN.ENABLED := TRUE
        END-IF
        #MITEM-DYN := #MITEM-DYN.SUCCESSOR
     END-REPEAT
   END-SUBROUTINE
*
   DEFINE SUBROUTINE IDLE
     PERFORM SWITCH-BARS
   END-SUBROUTINE
*
   DEFINE SUBROUTINE CLICK
     #CONTROL := *CONTROL
     IF #CONTROL.TYPE = MENUITEM AND #CONTROL.PARENT = #DIALOG.CONTEXT-MENU
          #MITEM-DYN := #CONTROL
          #CONTROL := #MITEM-DYN.CLIENT-HANDLE
          IF #CONTROL <> NULL-HANDLE
             IF #MITEM-DYN.CHECKED = CHECKED
                 #CONTROL.ENABLED := FALSE
                 #CONTROL.VISIBLE := FALSE
        ELSE
               #CONTROL.ENABLED := TRUE
               #CONTROL.VISIBLE := TRUE
        END-IF
      END-IF
     END-IF
   END-SUBROUTINE
*
  DEFINE SUBROUTINE BEFORE-OPEN
     #CTXMENU := #DIALOG.CONTEXT-MENU
     #MITEM-DYN := #CTXMENU.FIRST-CHILD
     REPEAT WHILE #MITEM-DYN <> NULL-HANDLE
        IF #MITEM-DYN.CLIENT-HANDLE <> NULL-HANDLE
          #CONTROL := #MITEM-DYN.CLIENT-HANDLE
        IF #CONTROL.VISIBLE
          #MITEM-DYN.CHECKED := CHECKED
      ELSE
          #MITEM-DYN.CHECKED := UNCHECKED
      END-IF
     END-IF
     #MITEM-DYN := #MITEM-DYN.SUCCESSOR
    END-REPEAT
   END-SUBROUTINE
```

```
*
   DEFINE SUBROUTINE AFTER-OPEN
     /* for MDI frames, unsuppress IDLE event to track active child change
     IF #DIALOG.TYPE = MDIFRAME
       #DIALOG.SUPPRESS-IDLE-EVENT := NOT-SUPPRESSED
     END-IF
     /* if dialog has no context menu, create one
     #CTXMENU := #DIALOG.CONTEXT-MENU
     IF #CTXMENU = NULL-HANDLE
         PROCESS GUI ACTION ADD WITH PARAMETERS
            HANDLE-VARIABLE = #CTXMENU
            TYPE = CONTEXTMENU
            PARENT = #DIALOG
     END-PARAMETERS GIVING *ERROR
     #DIALOG.CONTEXT-MENU := #CTXMENU
   END-IF
   /* unsuppress context menu's BEFORE-OPEN event for item update
   #CTXMENU.SUPPRESS-BEFORE-OPEN-EVENT := NOT-SUPPRESSED
   /* display bars according to context
   PERFORM SWITCH-BARS
  END-SUBROUTINE
*
  DEFINE SUBROUTINE SWITCH-BARS
     IF #DIALOG.TYPE = MDIFRAME
       #ACTIVE-DLG := #DIALOG.ACTIVE-CHILD
     END-IF
     IF #ACTIVE-DLG = NULL-HANDLE
       #ACTIVE-DLG := #DIALOG
     END-IF
     IF #ACTIVE-DLG <> #DIALOG.CLIENT-HANDLE
       #CTXMENU := #DIALOG.CONTEXT-MENU
       IF #CTXMENU <> NULL-HANDLE
         /* Remove any dynamic menu items previously created
         #CONTROL := #CTXMENU.FIRST-CHILD
         REPEAT WHILE #CONTROL <> NULL-HANDLE
           #MITEM-DYN := #CONTROL.SUCCESSOR
           IF #CONTROL.CLIENT-HANDLE <> NULL-HANDLE
              PROCESS GUI ACTION DELETE WITH #CONTROL
           END-IF
           #CONTROL := #MITEM-DYN
         END-REPEAT
         /* Search for all tool bar and dialog bar controls
         #CONTROL := #DIALOG.FOLLOWS
         REPEAT WHILE #CONTROL <> #DIALOG
            IF #CONTROL.TYPE = TOOLBARCTRL OR
                #CONTROL.TYPE = DIALOGBAR
              #CONTROL.CLIENT-KEY := 'CONTROL-ID'
              IF #CONTROL.CLIENT-VALUE = ' ' OR
                  #CONTROL.CLIENT-VALUE = #ACTIVE-DLG.NAME
               #CONTROL.VISIBLE := #CONTROL.ENABLED
               /* Create menu entry for bar
               PROCESS GUI ACTION ADD WITH PARAMETERS
                 HANDLE-VARIABLE = #MITEM-DYN
                 TYPE = MENUITEM
                 PARENT = #CTXMENU
                 STRING = #CONTROL.STRING
                 SUCCESSOR = #MITEM-DYN
                 CLIENT-HANDLE = #CONTROL
               END-PARAMETERS GIVING *ERROR
             ELSE
               #CONTROL.VISIBLE := FALSE
             END-IF
```

```
          END-IF
          #CONTROL := #CONTROL.FOLLOWS
        END-REPEAT
      END-IF
      /* Save handle of currently active dialog
      #DIALOG.CLIENT-HANDLE := #ACTIVE-DLG
    END-IF
  END-SUBROUTINE
END-SUBROUTINE
END
```