Dialogs

Dialogs are container widgets that provide a means of communicating between the user and the application. A dialog widget usually asks a question or presents some information to the user. In some cases, the application is suspended until the user provides a response.

Dialogs are similar to menus. Both seek input from the user. Like PopupMenus and PulldownMenus, dialogs appear in top−level windows and are more or less transient. Making a selection typically unposts a PopupMenu or PulldownMenu and often pops down a dialog. "PopupMenu" "in DialogBoxes" "PulldownMenu" "in DialogBoxes" There are two chief differences:

- Unless torn off, menus are usually *modal*; that is, the user must make a selection from the menu or unpost it before interacting with other parts of the application. Dialogs can be either modal or *modeless*. In a modeless dialog, the user can interact with other parts of the application before returning to the dialog.
- Menu components are limited to buttons, labels, and separators. Dialogs can contain other, sometimes arbitrary, kinds of widgets, such as List and Text. Dialogs permit more complex interaction with the user and allow the application to solicit a broader range of information.

Menus are well suited to allowing the user to make a single choice from a constrained set. Dialogs are appropriate for displaying information about a transient or unusual state of the program and for obtaining complex input from the user. Whether to use a dialog or a menu is not always clear. In fact, a TearOffMenu combines aspects of both. For more information on using menus and dialogs, see the *Motif Style Guide*.

BulletinBoard and DialogShell

From the application's point of view, a dialog must meet the following criteria:

- A dialog is a widget that is a subclass of **XmBulletinBoard**.
- A dialog's parent is a DialogShell.

BulletinBoard is intended to be the usual superclass for a dialog widget. The dialog widget can be either a BulletinBoard itself or one of its more specialized subclasses. BulletinBoard is a container with no automatically created children; it supplies general behavior needed by most dialogs. Its subclasses provide child widgets and specific behavior tailored to particular types of dialogs.

BulletinBoard and its subclasses can also function outside a DialogShell, as part of the application's main window. One subclass, Form, is particularly useful in providing constraint−based geometry management for a collection of child widgets.

BulletinBoard

BulletinBoard provides the following resources and behavior:

- Activation and cancellation of the dialog. BulletinBoard installs accelerators for `osfActivate` and `osfCancel`. Unless focus is in another button, `osfActivate` activates the **XmNdefaultButton** if it is sensitive. The `osfCancel` activates the **XmNcancelButton** if it is sensitive. Subclasses set the **XmNdefaultButton** and **XmNcancelButton**.
- A resource, **XmNdialogStyle**, that determines whether the dialog is modal or modeless. Three modal styles exist:

  primary application modal

  Among the dialog and its ancestors, input goes only to the dialog, but the user can interact with other parts of the application or with other applications.

  full application modal

  Within the application, input goes only to the dialog, but the user can interact with other applications.

  system modal

  Input goes only to the dialog; the user cannot interact with other applications. System modal should seldom be used.
- Callbacks invoked when the BulletinBoard is mapped and unmapped and when it gains input focus.
- Geometry−management resources and class methods that implement several resizing policies and that allow the BulletinBoard to interact with its subclasses in managing complex collections of descendant widgets. The geometry−related resources are **XmNmarginHeight**, **XmNmarginWidth** and **XmNresizePolicy**. For more information on BulletinBoard's geometry management, see Chapter 10.

Activation, Cancellation, and Help

Often a dialog has one or more actions, associated with buttons, that apply to the dialog as a whole. Some common actions are "activate," "cancel," and "help." BulletinBoard deals specially with activation and cancellation. BulletinBoard allows the user to "activate" or "cancel" the dialog from anywhere within the BulletinBoard (except, in the case of activation, when a button has the focus).

BulletinBoard has a resource, **XmNdefaultButton**, whose value is a button descendant that represents the default activation action. The dialog renders the default button so that it looks somewhat different from the other buttons. Use the **XmNdefaultButtonEmphasis** resource of **XmDisplay** to change the highlighting of this default button. When the user presses `osfActivate` in a button that has keyboard focus, that button's `osfActivate` actions are called. If the user presses `osfActivate` and no button has focus, BulletinBoard calls the `osfActivate` actions for **XmNdefaultButton** if it is sensitive. If the user presses `osfActivate` in a List, Text, or TextField descendant, the `osfActivate` actions for that widget are invoked first, and then BulletinBoard calls the `osfActivate` actions for **XmNdefaultButton**.

BulletinBoard has another resource, **XmNcancelButton**, whose value is a button descendant that represents the default cancellation action. When the user presses `osfCancel` anywhere within the BulletinBoard, BulletinBoard calls the `osfActivate` actions for **XmNcancelButton** if it is sensitive.

The help action works differently. Often the application represents help for the dialog as a whole by providing a Help button. When the user activates this button, the application provides help for the dialog. In general, the application can provide help through an **XmNactivateCallback** procedure for the Help button. Some BulletinBoard subclasses create Help buttons automatically. These widgets add a procedure to the Help button's **XmNactivateCallback** list that invokes the dialog's **XmNhelpCallback** procedures when the Help button is activated. In these cases, the application can provide help through the dialog's **XmNhelpCallback** procedures.

If the user presses `osfHelp` elsewhere in the BulletinBoard, this action usually invokes the **XmNhelpCallback** callbacks for the widget with the focus. If this widget has no **XmNhelpCallback** procedures, Motif looks up the widget hierarchy for the first ancestor with a non−NULL **XmNhelpCallback** list and invokes those procedures. By providing an **XmNhelpCallback** procedure for the dialog itself, the application can ensure that the user sees help for the dialog as a whole when the descendant widget with focus has no help information of its own.

DialogShell

DialogShell is the Motif shell widget that contains dialogs. It is a subclass of TransientShell, which is a subclass of VendorShell. DialogShell inherits much of VendorShell's behavior in interacting with the window manager and in providing geometry management for off−the−spot input methods.

DialogShell cooperates extensively with BulletinBoard, and some of DialogShell's features for containing dialogs assume that its child is a BulletinBoard or BulletinBoard subclass. Often the application does not need to deal directly with the DialogShell at all. The Motif convenience routines that create dialogs automatically create a DialogShell as the popup child of the parent shell.

To pop up the dialog, the application does not call **XtPopup** on the DialogShell, but instead manages the child of the DialogShell. DialogShell's *change_managed* procedure pops up the dialog when the child is managed and pops it down when the child is unmanaged, providing that the child's **XmNmappedWhenManaged** resource is True. If a BulletinBoard child's **XmNautoUnmanage** resource is initialized to True, the BulletinBoard is automatically unmanaged when its OK and cancel buttons are activated.

DialogShell notifies its BulletinBoard child by using the **XmNmapCallback** and **XmNunmapCallback** procedures when the child is about to be mapped and unmapped.

Like VendorShell, DialogShell ensures that when no off−the−spot input method exists the DialogShell window remains coincident with the child window. Setting **XmNx** and **XmNy** for the child sets these resources for the shell, without changing the child's position relative to the child. Setting **XmNheight**, **XmNwidth** and **XmNborderWidth** for the child usually sets these resources to the same value in the DialogShell. When a BulletinBoard child is managed with its **XmNdefaultPosition** resource set to True, DialogShell centers the dialog with respect to the parent.

BulletinBoard has two resources that allow the user or application to customize a parent DialogShell's interaction with the window manager. **XmNdialogTitle** provides a title for the window manager, and **XmNnoResize** determines whether or not the dialog MWM frame includes resize controls. To affect other aspects of interaction with the window manager, the user or application must set the appropriate DialogShell resources.

XmCreateBulletinBoardDialog creates a BulletinBoard and a parent DialogShell.

Initial Focus

When the **XmNkeyboardFocusPolicy** of a shell is **XmEXPLICIT**, Motif uses the Manager resource **XmNinitialFocus** in determining which component of a manager receives initial focus in these circumstances:

- When the manager is the child of a shell and the shell hierarchy receives focus for the first time
- When focus is inside the shell hierarchy, the manager is a composite tab group, and the user traverses to the manager using the keyboard

Following are the default values of **XmNinitialFocus** for BulletinBoard and its subclasses:

- For BulletinBoard, Form, and MessageBox, the default is the value of **XmNdefaultButton**
- For SelectionBox and its subclasses, the default is the text edit area

Making a Selection: SelectionBox

SelectionBox is a BulletinBoard subclass that generally allows the user to select an item from a list. By default, a SelectionBox includes the following children:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the list and text field
- Three or four buttons

The default buttons are OK, Cancel, and Help. By default, an Apply button is also created. If the parent of the SelectionBox is a DialogShell, it is managed; otherwise, it is unmanaged.

An application can add additional children to the SelectionBox. The first child is used as a work area. The value of **XmNchildPlacement** determines whether the work area is placed above or below the Text area, or above or below the List area. Additional children are laid out in the following manner:

MenuBar

The first MenuBar child is placed at the top of the window.

Buttons

All **XmPushButton** widgets or gadgets and their subclasses are placed after the OK button in the order of their creation.

Others

The layout of additional children that are not in these categories is undefined.

The user can select an item in two ways: by scrolling through the list and selecting the desired item or by entering the item name directly into the text edit area. Selecting an item from the list causes that item name to appear in the selection text edit area. SelectionBox installs accelerators, the value of **XmNtextAccelerators**, on the text edit widget. The default accelerators bind `osfUp`, `osfDown`, `osfBeginLine`, `osfEndLine`, and `osfRestore` events in the text edit widget to SelectionBox actions that select an item in the list and replace the text edit widget value with that list item.

SelectionBox provides **XmNokCallback**, **XmNcancelCallback**, **XmNhelpCallback**, and **XmNapplyCallback** lists, which the SelectionBox invokes when the corresponding button is activated. Activation of the OK button may invoke either the **XmNokCallback** list or the **XmNnoMatchCallback** list. When the user activates the OK button and either the **XmNmustMatch** resource is False or the text in the text edit area matches a list item, SelectionBox invokes the **XmNokCallback** procedures. When the user activates the OK button, **XmNmustMatch** is True, and the text in the text edit area does not match a list item, SelectionBox invokes the **XmNnoMatchCallback** procedures.

SelectionBox has two subclasses, FileSelectionBox and Command, which are described in later sections. **XmCreateSelectionDialog** creates a standard SelectionBox and a DialogShell parent. **XmCreatePromptDialog** creates a variant SelectionBox dialog containing a text edit area and label and OK, Cancel, and Help buttons. A PromptDialog has an unmanaged Apply button, and it has no List or List label. It is intended for the application to prompt the user for brief text input.

The **XmNdialogType** resource determines which of the standard SelectionBox children are created and managed. The value usually depends on the application's use of the SelectionBox:

- **XmDIALOG_SELECTION** usually indicates a standard SelectionBox dialog.
- **XmDIALOG_WORK_AREA** indicates a SelectionBox outside a DialogShell. The Apply button is unmanaged.
- **XmDIALOG_PROMPT** indicates a PromptDialog.
- **XmDIALOG_COMMAND** indicates a Command subclass.
- **XmDIALOG_FILE_SELECTION** indicates a FileSelectionBox subclass.

SelectionBox has resources for supplying text, label strings, and list items for its children. The widget IDs of the children of a SelectionBox and its subclasses are not available as resources. The application can retrieve the widget IDs of the automatically created children by using **XtNameToWidget** or by calling one of the convenience routines Motif provides for this purpose: **XmSelectionBoxGetChild**, **XmFileSelectionBoxGetChild**, and **XmCommandGetChild**.

Choosing a Pathname: FileSelectionBox

FileSelectionBox is a subclass of SelectionBox designed for finding and selecting files. By default, a FileSelectionBox contains the same children as a standard SelectionBox, with the addition of a second ScrolledList and a second text edit area. Unlike a SelectionBox, the default Apply button in a FileSelectionBox is labeled "Filter" in a C locale.

One of the text areas, the directory mask area, holds a directory mask specifying a base directory to be searched and a search pattern. However, if the **XmNpathMode** resource is set to **XmPATH_MODE_RELATIVE**, FileSelectionBox splits this text area into two separate TextFields. One of these TextFields holds the base directory to be searched and the other holds the search pattern.

Another text area, the selection area, holds the name of the selected file.

One of the Lists, the directory list, displays the subdirectories of the current base directory. The other List, the file list, displays all the files, subdirectories, or both in the base directory that match the search pattern. The **XmNfileListItems** resource holds the selected items in this list.

The **XmNfileFilterStyle** resource determines whether or not hidden files (those starting with a period, usually) will be displayed in the directory and file lists.

The user can select a new base directory to examine by scrolling through the list of directories and selecting the desired directory or by editing the directory mask. Selecting a new directory from the directory list does not change the search pattern. A user can select a new search pattern by editing the directory mask. Double clicking or pressing `osfActivate` on a directory in the directory list initiates a search for files and subdirectories in the new directory, using the current search pattern.

Activating the Filter button, the directory list, or the directory mask text area causes the FileSelectionBox to initiate a file search. The FileSelectionBox uses three procedures, each the value of a resource, in conducting the search: **XmNqualifySearchDataProc**, **XmNdirSearchProc** and **XmNfileSearchProc**. The **XmNqualifySearchDataProc** procedure extracts the base directory and the search pattern from the directory mask. The **XmNdirSearchProc** procedure uses the data returned by **XmNqualifySearchDataProc** to update the directory list. The **XmNfileSearchProc** procedure uses the data returned by **XmNqualifySearchDataProc** to update the file list.

The user can select a file by scrolling through the list of filenames and selecting the desired file or by entering the filename directly into the text edit area. Selecting a file from the list causes that filename to appear in the file selection text edit area. The user confirms the selection by activating the OK button, the file list, or the selection text area.

FileSelectionBox uses the SelectionBox callback lists to notify the application when the user activates one of the buttons. The application can also provide one or more of the three procedures that FileSelectionBox uses to conduct a search. For a specification of the input to and output from these routines, see the **XmFileSelectionBox**(3) reference page in the *Motif Programmer's Reference*.

The application can remove the directory list, the file list, or both. The application must unmanage the ScrolledWindow parent of the List and the corresponding label. An application can also add additional children to a FileSelectionBox, which manages any additional children in the same way as SelectionBox.

XmCreateFileSelectionDialog creates a FileSelectionBox and a parent DialogShell.

Command

Command is a SelectionBox subclass intended for entering a command. It contains the SelectionBox text edit area, List, and List label, but no buttons. The application can add only one additional work area child to the Command. A Command usually appears as part of the application's main window rather than as a dialog.

The user specifies a command by adding text to the text area or by selecting an item from the List, which represents the command history. Whenever the text edit area changes, Command invokes the **XmNcommandChangedCallback** procedures. The user enters a command by activating the List or the text edit area. When the user enters a command, Command appends the command to the history list and invokes the **XmNcommandEnteredCallback** procedures.

Command has a number of resources that are aliases for SelectionBox resources dealing with the List and text edit area. Command also has an **XmNhistoryMaxItems** resource, which specifies the maximum length of the history list. After the list reaches this length, Command deletes the first item in the list before appending a newly entered command.

MessageBox

MessageBox is a BulletinBoard subclass intended for a dialog consisting of a single user interaction. By default, a MessageBox has the following components:

- A LabelGadget with a pixmap label symbolizing the type of interaction the MessageBox represents
- A LabelGadget with a compound string label representing the text of the message
- A SeparatorGadget separating the message symbol and text from the other children
- Three buttons: OK, Cancel, and Help

Typically the message symbol and text are on top and the buttons on the bottom, with the separator between. The application can add additional children to a MessageBox. Additional children are laid out in the following manner:

- The first MenuBar child is placed at the top of the window.
- All **XmPushButton** widgets or gadgets, and their subclasses, are placed after the OK button in the order of their creation.
- A child that is not in these categories is treated as a work area and is placed above the row of buttons. If a message label exists, the child is placed below the label. If a message pixmap exists, but a message label is absent, the child is placed on the same row as the pixmap. The child behaves as a work area and grows or shrinks to fill the space above the row of buttons. The layout of multiple work area children is undefined.

Several convenience routines create MessageBox widgets with DialogShell parents for particular kinds of interactions. For most of these routines, the principal difference in the type of MessageBox they create is that each uses a distinct default symbol pixmap. When it creates the symbol pixmap, MessageBox uses **XmGetPixmapByDepth** to find a pixmap with a name that corresponds to the type of interaction. Each dialog type is also associated with a value of the **XmNdialogType** resource. The following table shows the correspondence between creation routine, **XmNdialogType**, and symbol pixmap name:

Table 1 MessageBox Routines, Dialog Types, and Pixmaps

Convenience Routine	**XmNdialogType**	*Pixmap Name*
XmCreateErrorDialog	**XmDIALOG_ERROR**	*xm_error*
XmCreateInformationDialog	**XmDIALOG_INFORMATION**	*xm_information*
XmCreateMessageDialog	**XmDIALOG_MESSAGE**	
XmCreateQuestionDialog	**XmDIALOG_QUESTION**	*xm_question*
XmCreateTemplateDialog	**XmDIALOG_TEMPLATE**	
XmCreateWarningDialog	**XmDIALOG_WARNING**	*xm_warning*
XmCreateWorkingDialog	**XmDIALOG_WORKING**	*xm_working*

A MesssageDialog and a TemplateDialog have no default symbol pixmap. A TemplateDialog is a special MessageBox variant that is intended for application customization and that, by default, has no children except the separator.

Like SelectionBox, MessageBox has **XmNokCallback**, **XmNcancelCallback**, and **XmNhelpCallback** lists to inform the application when the user activates a button. MessageBox has resources for supplying label strings and the symbol pixmap for its children. The widget IDs of the children of a MessageBox are not available as resources. The application can retrieve the widget IDs of the automatically created children by using **XtNameToWidget** or by calling **XmMessageBoxGetChild**.

Form

Form is a BulletinBoard subclass whose main purpose is to provide constraint−based geometry management for arbitrary children. Form has a number of constraint resources that it uses to place children with respect to the Form, positions within the form, and other children. Most Form −specific behavior is related to this geometry management. Form has no default children of its own. But as a BulletinBoard subclass, Form is an appropriate container for use in dialogs. **XmCreateFormDialog** creates a Form and a DialogShell parent.

For information on Form's geometry management, see Chapter 14.