Common Desktop Environment (CDE) 5.2

Application Builder User's Guide

Document Number 860-0202-003

RECORD OF REVISION

Version	Description
001	June 2002. Common Desktop Environment 5.2.

Common Desktop Environment (CDE) 5.2 Application Builder User's Guide
Document Number 860-0202-003

Contents

Preface

This manual introduces App Builder and shows you how best to use it. See “Overview of the App Builder Process” on page 4 for a summary description.

Who Should Use This Book

This user’s guide is for anyone who wants to build or prototype a user interface with App Builder. Because you can easily create and modify user interfaces without writing any code using App Builder, it is a powerful tool for programmers and non-programmers—including user interface designers and project managers.

How This Book Is Organized

Chapter 1, “Getting Started,” includes an annotated picture of the App Builder primary window, an overview of the process of building an application, and instructions for starting App Builder.

Chapter 2, “Managing Projects and Modules,” explains how to create, open, save, edit, and close projects and modules.

Chapter 3, “Laying Out a User Interface,” explains how to drag and drop objects from the object palettes and how to align and distribute control objects once they are dropped.

Chapter 4, “Editing Properties of Interface Objects,” explains how to edit object properties in the Revolving Property Editor.

Chapter 5, “Creating and Editing Panes, Menus, and Messages,” explains how to create pane objects, menus, and message dialog boxes.

Chapter 6, “Adding Functionality to the Interface,” explains how to create on-item help, how to create functional connections between objects, and how to establish drag and drop and application framework behavior.

Chapter 7, “Grouping and Attaching Objects,” explains how to group control objects and how to attach objects to each other for dynamic resize behavior.

Chapter 8, “Testing Menus, Help, and Connections,” explains how to change to test mode for testing certain App Builder functions.

Chapter 9, “Generating Code and Building an Application,” describes the Code Generator and explains how to generate code, make your application, and run it.

Appendix A, “App Builder Windows and Dialog Boxes,” describes the primary window, including its object palettes and menus, and other App Builder windows, including the Project Organizer, the Module Browser, and the Code Generator.

Appendix B, “Revolving Property Editor,” describes the Revolving Property Editor in general and each of the individual property editors specifically.

What Typographic Changes and Symbols Mean

The following table describes the typefaces and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`Monospace`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail`.
Italic	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
Italic	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
Code samples are included in boxes and may display the following:		
`%`	UNIX C shell prompt	`%`
`$`	UNIX Bourne and Korn shell prompt	`$`
`#`	Superuser prompt, all shells	`#`

Getting Started 1

The Application Builder *(*referred to throughout this manual as App Builder*)* is a development tool that makes designing, creating, and prototyping a user interface easier. App Builder gives you the freedom to create and try user interfaces without writing any code. Because you can create and modify an interface easily, you'll find that you can spend more time designing and testing, the surest route to better user interfaces.

Application Builder Primary Window

The Application Builder primary window, shown below, is the starting point for creating a user interface. See Appendix A, “App Builder Windows and Dialog Boxes,” for a detailed description of the primary window, including each of its menus.

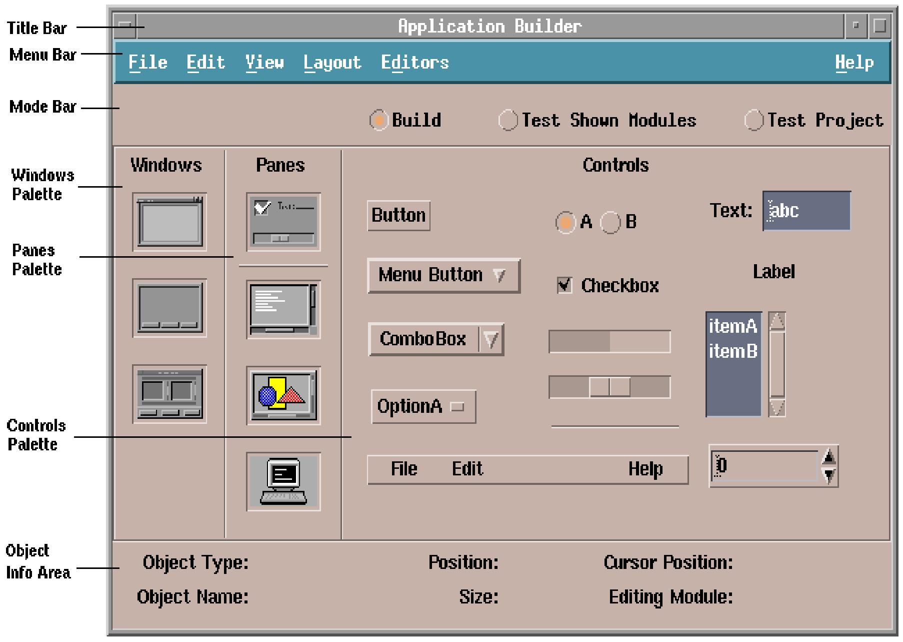


Figure 1-1 Application Builder primary window

The basic method for creating an App Builder user interface is to *drag and drop* objects from the App Builder primary window onto the desktop or onto other App Builder objects. See Chapter 3, “Laying Out a User Interface,” for details.

Starting and Exiting App Builder

▼ To Open App Builder from an Icon

- ♦ If App Builder has previously been open and the App Builder icon is on the desktop, double-click the icon to open App Builder.
- ♦ If App Builder is installed on the Front Panel, click the App Builder icon in the Personal Applications subpanel to open App Builder.

  To install App Builder on the Front Panel, see "To Put an Application Icon in the Front Panel" in the Application Manager help volume for instructions.

▼ To Start App Builder from the Command Line

The command to run App Builder is `dtbuilder`. Do the following to start App Builder from the command line:

- ♦ Type `dtbuilder`

  If `dtbuilder` is in your path, App Builder will start. If it is not in your path, you will need to type the full path name (which, by default, is `/usr/dt/bin/dtbuilder`) or change to the folder where `dtbuilder` is located before typing `dtbuilder`.

▼ To Exit App Builder

- ♦ Choose Exit from the File menu of the App Builder primary window to quit App Builder.

  If you have not saved all changes, a message dialog box will be displayed, giving you the opportunity to discard the changes and continue the exit process or to cancel the exit process and continue running App Builder. Click Discard Changes if you do not want to save them. Click Cancel if you do not want to discard your changes; you could then save your changes and exit.

Overview of the App Builder Process

The basic process of building and maintaining a user interface with App Builder is simple and straightforward:

1. Start App Builder.
2. Open a new project and a new module.
3. Drag and drop windows (main windows and custom dialogs) to the desktop, creating a new module for each window.
4. Drag and drop panes onto main windows or custom dialogs.
5. Drag and drop controls (buttons, choice objects, text fields, for example) onto control panes.
6. Create pane objects, menus, help dialogs, and message dialogs.
7. Edit the properties of interface objects.
8. Make functional connections between objects in the user interface.
9. Go into test mode to test menus, help, and connections.
10. Display the Code Generator to generate code and make the user interface.
11. Add user code to the code generated by App Builder.
12. Debug the code, make and run the application.
13. Repeat the process to modify and maintain the user interface.

There are many variations on this formula, but the process is similar for any application.

Object Types

There are three basic types of objects on the primary window: windows, panes, and controls. See Appendix A, “App Builder Windows and Dialog Boxes,” for descriptions of each of the objects.

The windows in App Builder are:

- Main window
- Custom dialog
- File selection dialog

The panes in App Builder are:

- Control pane
- Text pane
- Draw area pane
- Term pane

The controls in App Builder are:

- Button
- Menu button
- Combo box
- Option menu
- Menu bar
- Radio box
- Check box
- Gauge
- Scale
- Separator
- Text Field
- Label
- List (scrolling list)
- Spin box

Rules for Dropping Objects

The rules for dragging and dropping the three types of App Builder objects are explained below. An error message will be displayed if you attempt to drop an object on an illegal target.

Windows (main window, custom dialog, file selection dialog) can be dropped anywhere on the desktop except for the App Builder primary window.

Panes (control pane, text pane, draw area pane, term pane) can be dropped on a main window, a custom dialog, or on another pane. See “Creating and Editing Pane Entities” on page 38 for more information.

Controls (buttons, menus, boxes, for example) can be dropped on a control pane or a group.

Managing Projects and Modules 2

When you use App Builder to create a graphical user interface, you are working on a *project*, which is comprised of one or more *modules*. App Builder, which was built with itself, was a single project comprised of over 30 modules.

Creating, Opening, and Saving Projects

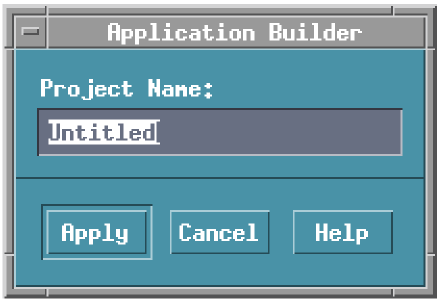
A project file is started when you choose New Project from the File menu of the App Builder primary window or New from the Project menu of the Project Organizer, or when you drag and drop a window onto the desktop in a new session of App Builder.

A project file is saved when you choose Save Project from the File menu of the App Builder primary window or when you select Save from the Project menu of the Project Organizer. A project file has a `.bip` (builder interface project) suffix.

▼ To Create a New Project

1. Choose New Project from the File menu.

   The Project Name dialog box will be displayed.

   If you have made changes to the current project since you last saved it, a message dialog box will be displayed first, giving you the option to discard the changes and create the new project or to cancel the New Project operation.

   Click Discard to throw out the changes and close the current project.

   Click Cancel if you want to save the current project. Save the current project before creating the new project.

2. Type a name (all lowercase) for the project and click Apply.

   The name of the project (with `.bip` added as a suffix) will be displayed in the title bar at the top of the App Builder primary window. Every module you create or import will be part of the current project until you open another project.

Note – Project names should be all lowercase so that there is no conflict between the name of the project resource file and the project executable file. The name of the resource file created when you generate code is the same as the name of the project, minus the `.bip` suffix, but it is given an initial capital letter.

By default, an unnamed project is called `Untitled.bip`. You can give the project a different name when you save it.

▼ To Open an Existing Project

1. Choose Open Project from the File menu of the App Builder primary window or Open from the Project menu of the Project Organizer.

   The Open Project dialog box will be displayed.

   If you have made changes to the current project since you last saved it, a message dialog box will be displayed first, giving you the option to discard the changes and open the other project or to cancel the Open Project operation.

   Click Discard to throw out the changes and close the current project.

   Click Cancel if you want to save the current project. Save the current project before opening the other project.

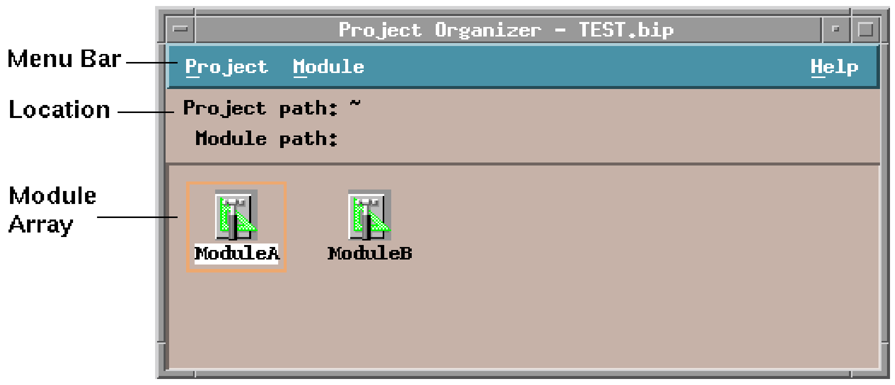
2. Change folders, if necessary.

   You have to press Return or click Update before the folder change is registered.

3. Double-click the appropriate project file (`.bip` suffix) in the Files list

   Or, select the file and click Open.

   The name of the project will be displayed in the title bar of the App Builder primary window and the selected project will be displayed in the Project Organizer.



4. Select the modules you want to display and choose Show from the Module menu to display the module interfaces. See “To Show a Hidden Module” on page 18 for detailed instructions.

▼ To Save a Project

A project is only saved when you explicitly choose to save it, so be sure to save often and regularly.

1. Choose Save Project from the File menu of the App Builder primary window or Save from the Project menu of the Project Organizer.

   If you have saved the project before, the project will be saved without comment.

   If this is the first time you have saved the project, the Save Project dialog box will be displayed.

2. Change to the appropriate folder.

   You will normally want a separate folder for each project you work on.

3. Type a file name in the Enter file name field.

   You do not have to append `.bip` to the project name; this is done automatically when you save a project.

4. Click Save.

   The project will be saved.

▼ To Rename a Project

Do the following to save the current project with a different name. See “To Save a Version of a Project” below if you want to save a *version* of the current project.

1. Choose Save Project As from the File menu of the App Builder primary window or Save As from the Project menu of the Project Organizer.

   The Save Project dialog box will be displayed, with the current project name in the Enter file name field.

2. Modify the name or type in a new name in the Enter file name field.

3. Click Save.

   A message dialog box will be displayed for each module in the project, telling you that the module exists and giving you the option to overwrite it or cancel the operation.

4. Click Overwrite for each module if you want to rename the project and save the module.

   Click Cancel if you do not want to rename the project and overwrite the current module.

   If you click Overwrite for each of the modules the project will be renamed; the new project name will be displayed in the title bar of the App Builder primary window. The old project file (`.bip` suffix) will still be in the folder, but it will not be the active project file. If you generate code for the project and run `make` in the folder, the new project name will be used.

▼ To Save a Version of a Project

Do the following to save a version of the current project in a different folder. Note that if you have made unsaved changes to the current project those changes will be saved in the new project only.

1. Choose Save Project As from the File menu of the App Builder primary window or Save As from the Project menu of the Project Organizer.

   The Save Project dialog box will be displayed, with the current project name in the Enter file name field.

2. Change to another folder.

3. Type a name in the Enter file name field.

4. Click Save.

   The project—the project file (`.bip` suffix) and all of the module files (`.bil` suffixes)— has been copied to another folder. The original project and module files are not affected. The new project will now be the current project; its name will be displayed in the title bar of the App Builder primary window.

▼ To Save a Project to a File (Encapsulate Project)

A project is comprised of one or more modules. Normally a project file is saved in a file with a `.bip` suffix, and each module file is saved in a separate file with a `.bil` suffix. To save a project as a single file (for convenience in mailing the project to someone, for instance):

1. Open the project, as described in "To Open an Existing Project" on page 9.
2. Choose Save Project As from the File menu of the App Builder primary window or Save As from the Project menu of the Project Organizer.
3. Change to the appropriate folder, if necessary.
4. Select Save As Encapsulated Project.

   The name of the current project will be displayed in the Enter file name field, with a `.bix` (builder interface exchange) suffix.
5. Click Save or press Return.

Note – When a project is saved as an encapsulated file, the `.bip` file is not affected. When an encapsulated project is opened in App Builder, it is opened just like any other project. When you attempt to save a project that was opened from an encapsulated file, a message dialog box will be displayed, explaining that the project will be saved as individual files. Choose Save Project As if you want to save it as an encapsulated project again.

▼ To Close a Project

♦ Choose Close Project from the File menu of the App Builder primary window or Close from the Project menu of the Project Organizer.

If you have made changes since saving the project a message dialog box will be displayed, giving you the chance to discard the changes or to cancel the close operation.

Creating, Importing, Exporting, and Saving Modules

A *module* is a logical unit of a project. Each window and dialog in App Builder is a module of the App Builder project. For instance a module is created when you choose New Module from the File menu of the App Builder primary window or New from the Module menu of the Project Organizer.

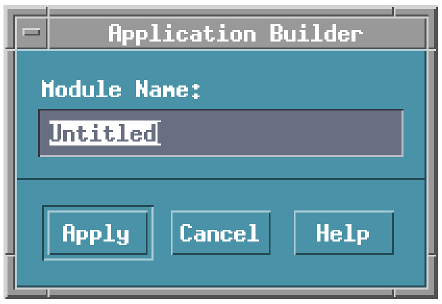
All module files in a project are saved when the project is saved. You can explicitly save a particular module by choosing Save from the Module menu of the Project Organizer. A saved module file has a `.bil` (builder interface language) suffix.

▼ To Create a New Module

To create a new module, which will become part of the current project:

1. Choose New Module from the File menu of the App Builder primary window or New from the Module menu of the Project Organizer.

   The Module Name dialog box will be displayed, with Untitled selected as the default name:



Note – If you drag and drop a window on the desktop after creating a new project, the Module Name dialog box will be displayed, just as if you had chosen New Module from the File menu.

2. In the dialog box, type in the name you want to give the new module.

3. Click Apply or press Return.

   The name of the new module will appear in the Editing Module field at the bottom of the App Builder primary window. Any windows you drag from the Windows palette and drop on the desktop will be part of the new module.

▼ To Import a Module into a Project

To import an existing module into the current project:

1. Choose Import Module from the File menu of the App Builder primary window or Import from the Module menu of the Project Organizer.

   The Import File dialog box will be displayed.

2. Change to the folder where the module (`.bil` suffix) file is saved.
3. Change the Import Format type, if necessary.

   By default, BIL format is selected. If the file you are importing is a UIL file, click the UIL button. The file will be converted to BIL format when it is imported.

4. Change Import By method, if necessary.

   By default, Import By Copy is selected. If you want to import the module by reference rather than making a copy of it, click the Reference button.

Note – Import By Reference, which causes module files to be shared, can be dangerous, since the actual module file may be changed or deleted inadvertently.

---

5. Double-click on the module to be imported in the Files list.

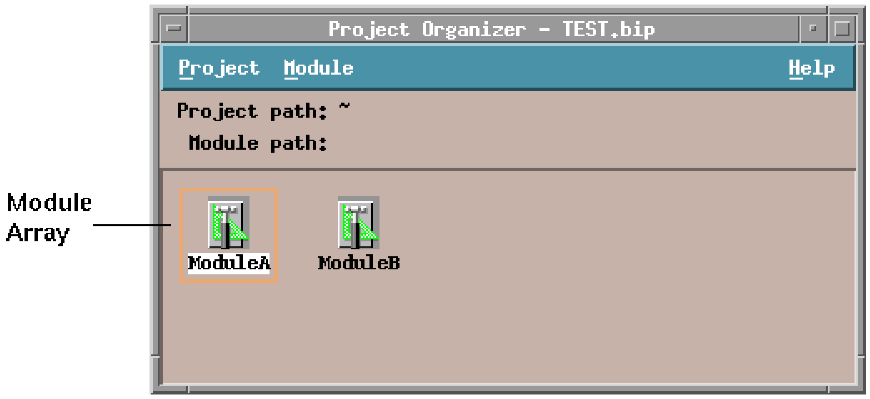
   Or, select the file and click Import.

   The module will be added to the current project the next time you save the project.

▼ To Save a Module

All modules in a project are saved when you save the project. If you want to save individual modules, you can do so in the Project Organizer.

1. Display the Project Organizer by choosing Project Organizer from the File menu of the App Builder primary window.



Project path: ~
Module path:

2. In the module array of the Project Organizer select the module you want to save.
3. Choose Save from the Module menu.

   If you have saved the module previously during this App Builder session, the module will be saved without comment.

   If this is the first time you have saved the module, the Save BIL File dialog box will be displayed, with the name of the selected module (with a `.bil` suffix) in the Enter file name field.
4. Change to the folder where you want to save the module, if necessary.
5. Click Save or press Return.

▼ To Rename a Module

Use Save As from the Module menu of the Project Organizer to rename a module. When you save the current project, the new module name will replace the old name in the project (`.bip`) file. The original module will still be in the project folder, but it will not be part of the project. To save a module without affecting the project, see "To Export a Module" on page 16.

1. Display the Project Organizer by choosing Project Organizer from the File menu of the App Builder primary window.
2. Select the module you want to rename.
3. Choose Save As from the Module menu.

   The Save BIL File dialog box will be displayed, with the name of the selected module (with a `.bil` suffix) in the Enter file name field.
4. Type a file name in the Enter file name field.
5. Click Save or press Return.

   The new module name will replace the old name in the project (`.bip`) file the next time you save the project.

▼ To Export a Module

To save a version of a module in the current project folder or to save a module separately from the current project:

1. From the File menu of the App Builder primary window choose Export Module and select one of the currently open modules from the submenu displayed.

   Or, in the Project Organizer select the module to be exported in the module array and choose Export from the Module menu.

   The Export File dialog box will be displayed, with the selected module name in the Enter file name field.

2. Type a new file name in the Enter file name field.

   Or, change to the folder where you want to save the module and type a file name in the Enter file name field.

   If you want to save a version of the module in the current folder, do not change folders. Simply give the module a different name.

3. Click Export or press Return.

Note – The current project is not affected when you export a module. The module is not part of the current project.

▼ To Save a Module in UIL Format

To save a module in UIL (user interface language) format instead of BIL (builder interface language) format:

1. Choose Export Module from the File menu of the App Builder primary window and select the module you want to export from the submenu that is displayed.

   Or, in the Project Organizer select the module to be exported and choose Export from the Module menu.

   The Export File dialog box will be displayed, with the selected module name in the Enter file name field.

2. Change to the folder where you want to save the module, if necessary.

3. Select Save As UIL (above the Enter file name field).

   The file name suffix will change from `.bil` to `.uil`.

4. Type a file name in the Enter file name field, if necessary.

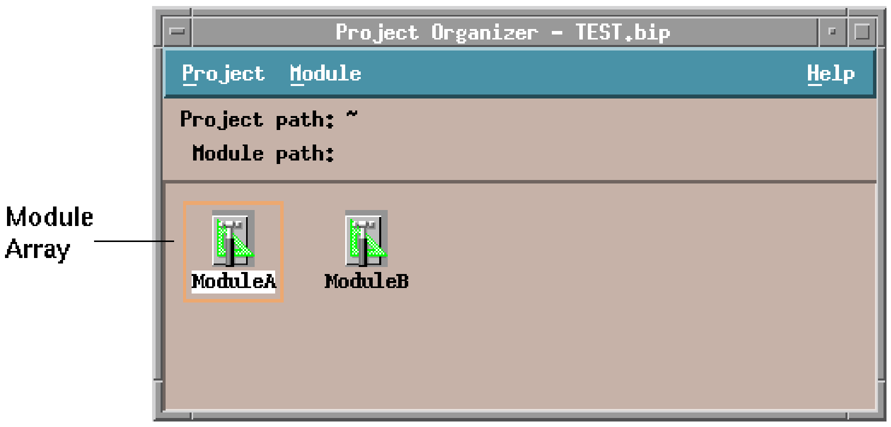
   If the name in the Enter file name field is OK, leave it as it is.

5. Click Export or press Return.

   The file will be saved with a `.uil` suffix.

Showing, Hiding, and Removing Modules

For a small project you may always want to show all modules. For a large project with many modules you may want to show only one or two modules at a time. Use the Project Organizer to show and hide modules, and to remove modules from projects.



▼ To Show a Hidden Module

The Project Organizer displays icons for all of the modules that comprise a project. You can show or hide the interfaces for any of the modules in the project.

1. Display the Project Organizer by choosing Project Organizer from the File menu of the App Builder primary window.
2. Double-click the module icons in the module array of the Project Organizer that you want to show

   Or, select the module icons and choose Show from the Module menu.

   The user interfaces for the selected modules will be displayed.

Note – If a module you want to show is in a different project, you will first have to open the other project. See "To Open an Existing Project" on page 9 for instructions.

▼ To Hide a Shown Module

To hide a module that is displayed (to clean up the desktop so that you can more easily work on another module, for instance):

1. Display the Project Organizer by choosing Project Organizer from the File menu of the App Builder primary window.
2. Select the modules in the module array that you want to hide.

   Select one module by clicking mouse button 1 on it. To add to the selection click mouse button 2 on other modules. To select a number of adjacent modules drag-select with mouse button 1 or mouse button 2, starting above and to the left of the first module to be selected.
3. Choose Hide from the Module menu.

   The user interfaces for the selected modules will be hidden.

▼ To Remove a Module from a Project

When you remove a module from the current project, the module file will still exist in the project folder, but it will no longer be part of the project. The module file name will be removed from the project file (`.bip` suffix) the next time the project is saved.

1. Display the Project Organizer by choosing Project Organizer from the File menu of the App Builder primary window.
2. Open the project that contains the module to be removed, if necessary.

   Since you can have only one project open at one time, you will have to close the current project first. See "To Open an Existing Project" on page 9 for instructions.
3. Select the modules in the module array that you want to remove.

4. Choose Remove from the Module menu of the Project Organizer.

Laying Out a User Interface 3

The basic App Builder process for laying out an interface is to drag objects from the App Builder primary window and drop them on the desktop or on other App Builder objects.

See "Application Builder Primary Window" on page 2 for an annotated picture of the primary window. See Appendix A for a full description of the primary window and its elements.

Dragging and Dropping Palette Objects

The rules for dropping palette objects are simple; they are enforced by error messages when they are violated.

- Windows (main window, custom dialog, file selection dialog) are dropped on the desktop.
- Panes (control, draw area, text, and term) are dropped on windows or on other panes.
- Controls (buttons, boxes, choice objects, and others) are dropped on a control pane.

▼ To Create a Main Window or Custom Dialog

1. Drag a main window or a custom dialog from the Windows palette and drop it on the desktop.

   The module name will be displayed in the status area at the bottom of the window.

   If you haven't previously named the module, the Module Name dialog box will be displayed.

2. Move the cursor to the Module Name dialog box, type a name, and click Apply, if necessary.

   The module name will be displayed in the status area at the bottom of the window.

3. Edit the properties of the main window or dialog, if necessary.

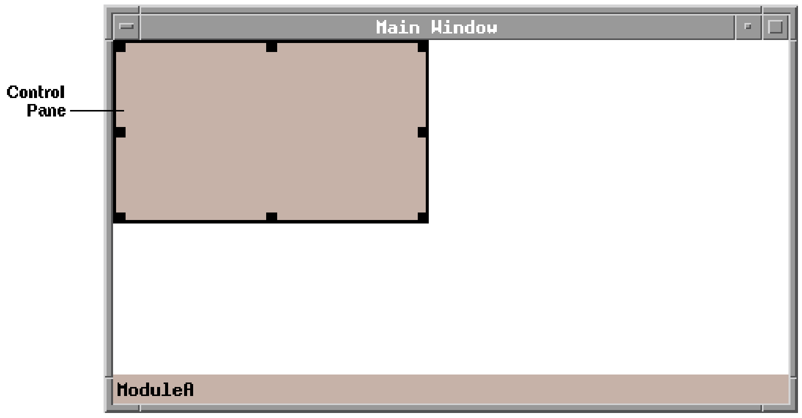
   This can be done now or later. See "To Edit the Properties of an Object" on page 30 for general instructions. See"To Edit Properties of a Main Window or a Primary Main Window" on page 33 for specific instructions for a main window or a primary main window.

▼ To Create a Window with a Spanning Control Pane

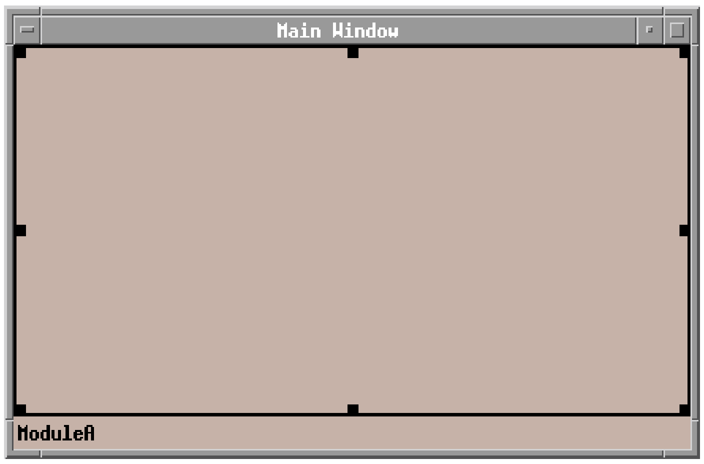
Often you will want a control pane to fill the entire blank pane area of a main window or custom dialog. You can then drop controls or other panes on the control pane to create a complex window such as the App Builder primary window. Do the following once you have dropped a main window or custom dialog on the desktop.

Note – Once a control pane has been resized to cover the blank pane area, the main window or custom dialog object can only be selected in the status region at the bottom of the window (or in the Module Browser).

1. Drag a control pane from the Panes palette and drop it on the top-left corner of the main window or custom dialog.



2. Drag the bottom-right corner of the control pane (an arrow pointing towards a corner will be displayed) beyond the bottom-right corner of the window.



The control pane will be attached to the four sides of the window. If you resize the window, the control pane will be resized with it.

See “To Attach Objects in an Interface” on page 95 for details about attachments.

Selecting and Aligning Interface Objects

For many actions, including moving, aligning, and grouping, you need to select one or more control objects in an interface. You can only multiply-select *siblings*—objects that are the children of the same parent. (All control objects in a single control pane are siblings, for instance.) Any panes that are dropped on a control pane and created as children of the control pane are also siblings of controls in the control pane.

▼ To Select Control Objects

- Select one object by clicking it in the interface or in the Module Browser.

  Selecting an object in the Module Browser also selects it in the interface, and vice versa.
- Select a number of adjacent objects by positioning the mouse cursor above and to the left of the objects, pressing mouse button 1, and dragging the mouse to encompass other objects down and to the right of the first object.
- Add or subtract an object to the current selection by clicking mouse button 2 on the object.

  If an object is selected, clicking mouse button 2 on it unselects it.
- To add a number of adjacent objects to those that are selected, position the mouse cursor above and to the left of the objects to be added, press mouse button 2, and drag the mouse to encompass other objects down and to the right of the first object.
- To deselect all but one object, click mouse button 1 on an object.

  Only that object will be selected.

Note – When you have selected a number of objects in the interface, all the objects will move if you press mouse button 1 on one of the objects and move the mouse. A rectangular border will be drawn around the objects as you move the mouse.

▼ To Align Objects in an Interface

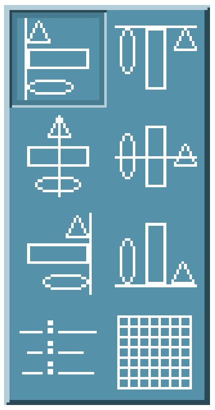
This procedure describes “static” alignment of objects: the objects are aligned one time only. See Chapter 7, “Grouping and Attaching Objects,” for instructions to find out how to group and attach objects for “dynamic” alignment.

1. Select two or more objects.

   See “To Select Control Objects” on page 25 for instructions.

2. Choose Align from the interface pop-up menu (displayed by pressing mouse button 3) and select one of the alignment icons from the submenu.

   The selected objects will be aligned according to the alignment choice. Choices are described below. Vertical alignment icons are on the left and are described first.



- Left-edge: Aligns selected objects vertically along their left edges.
- Vertical-center: Aligns selected objects vertically on their horizontal centers.
- Right-edge: Aligns selected objects vertically along their right edges.
- Colon: Aligns selected objects vertically along their colons or labels.
- Top-edge: Aligns selected objects horizontally along their top edges.
- Horizontal-center: Aligns selected objects horizontally on their vertical centers.
- Bottom-edge: Aligns selected objects horizontally along their bottom edges.
- Grid: Does no alignment at this time.

Note – If you select objects that are arranged horizontally and choose a vertical alignment (or vice versa), the objects will end up on top of one another. You can unstack the objects by choosing Distribute from the pop-up menu immediately after the align function (the objects will still be selected). See “To Distribute Objects Evenly” on page 27 for instructions.

▼ To Distribute Objects Evenly

This procedure describes “static” adjustment of the spacing between objects: the spacing is adjusted one time only. See Chapter 7, “Grouping and Attaching Objects,” for instructions to find out how to group and attach objects for “dynamic” spacing alignment.

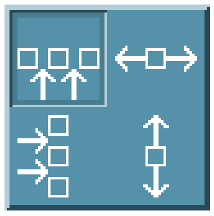
1. Select one or more objects.

   See “To Select Control Objects” on page 25 for instructions. Select one object to center it between the edges of its parent.

2. Choose Distribute from the interface pop-up menu (displayed by pressing mouse button 3) and select one of the distribute icons from the submenu.

   The selected objects will be distributed or centered according to your choice.

   Objects will be spaced 10 pixels apart horizontally or vertically if you choose one of the distribute choices. If you choose one of the centering choices, the object or objects will be centered within the boundaries of the parent control pane.



- Horizontal-space: Distributes selected objects horizontally 10 pixels apart. The left-most object is the anchored object, which does not move.
- Vertical-space: Distributes selected objects vertically 10 pixels apart. The top-most object is the anchored object, which does not move.
- Horizontal-center: Centers selected objects horizontally between the left and right edges of the parent object, maintaining the distance between selected objects.
- Vertical-center: Centers selected objects vertically between the top and bottom edges of the parent object, maintaining the distance between selected objects.

Editing Properties of Interface Objects 4

Once you have dropped an object or have created an object in the interface, you will want to customize the object by editing it in the Revolving Property Editor. See Appendix B, “Revolving Property Editor,” for an illustration of a property editor and descriptions of each of the elements in all of the property editors.

All objects dragged from the App Builder palettes have *properties* that can be edited. These properties include object name, color, and a variety of other characteristics, depending on the object type.

▼ To Open a Property Editor

- Double-click an object in the interface or in the Module Browser to open the Revolving Property Editor with the clicked-on object selected.
- *Or*, select an object in the interface or in the Module Browser and choose Props (Revolving or Fixed) from the pop-up menu (displayed by pressing mouse button 3 in the interface or the Browser) to open the property editor with the object selected.
- *Or*, choose Properties from the Editors menu in the App Builder primary window.

  The Revolving Property Editor will be displayed, with the object most recently selected in the interface or the Module Browser selected in the Revolving Property Editor.

▼ To Edit the Properties of an Object

Once you have displayed the property editor, do the following to edit the properties of an object:

1. Choose the object type that you want to edit from the Object Type menu at the top of the Revolving Property Editor, if necessary.

   If you double-clicked an object to display the Revolving Property Editor or if the object was selected when you chose Props from one of the pop-up menus, the object type and the specific object will already be selected.

   If a tear-off (Fixed) editor is displayed, there is no Object Type menu.

2. Select the object that you want to edit in the Objects scrolling list, if necessary.

   The object may already be selected.

3. Modify any of the properties, as appropriate.

   See Appendix B, “Revolving Property Editor,” for descriptions of each of the elements of the property editors.

Note – The easiest way to perform item editing in those property editors that have an item list is to select the first item in the list, thus selecting it in the text field. Type a new name for the item and click the OK button. The new name will be displayed in the item list and the next item in the list will be selected. Continue down the list with this select, type, Return process until all items are completed. Property editors with item lists include the choice objects (Radio Box, Check Box, Option Menu), Combo Box, List, Menu, Menubar, and Spin Box.

4. Click the Apply button to apply the changes and leave the property editor displayed.

   Click the OK button to apply the changes and close the property editor.

   Click Reset to reset all changed elements to their values at the last Apply.

   Click Cancel to reset all elements to their values at the last Apply and close the property editor.

See “To Edit Properties of a Main Window or a Primary Main Window” on page 33 for specific instructions for editing the properties of a main window.

▼ To Display a Fixed Property Editor

The Revolving Property Editor is a single dialog box that displays one of 20 property editors, depending on the item you choose from the Object Type option menu. To display a separate, fixed property editor of a specific object type:

1. Select the object you want to edit in the interface or in the Module Browser.
2. Choose Props from the pop-up menu (displayed by pressing mouse button 3 in the interface or in the Module Browser) and select Fixed from the Props submenu.

   A fixed version of the property editor for the selected object type will be displayed.

Or,

1. Choose the object type you want to edit in the Object Type menu of the Revolving Property Editor.

2. Click the Tear-off button at the top-right of the Revolving Property Editor.

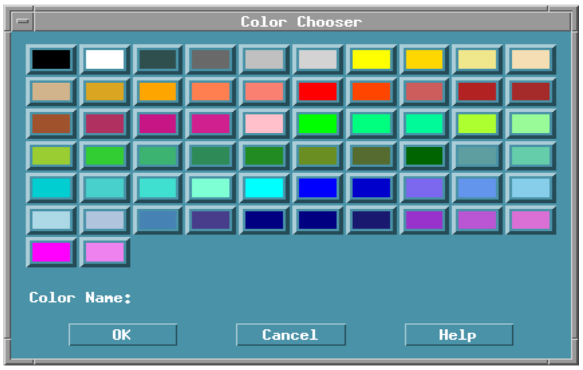
   A fixed version of the property editor for the selected object type will be displayed.

▼ To Select Colors from the Color Chooser

Most property editors have background and foreground color properties. If you know the name of the color you want to use, type it in the text field next to Color:Background or Color:Foreground. To select a color from the Color Chooser palette:

1. Click the Background or Foreground menu button and choose Color Chooser.

   The Color Chooser, with an array of color choices available, will be displayed.



2. Click the desired color in the palette.

   The name of the selected color will be displayed after Color Name.

3. Click OK to select the color and dismiss the Color Chooser.

   The selected color will be displayed in the rectangle next to the Background or Foreground menu and the name of the color will be displayed in the text field next to the colored rectangle.

4. Repeat the process for Background or Foreground, if desired.

5. Click Apply in the property editor to apply the changes.

   The background or foreground of the object in the interface will display the selected color.

Main Windows and the Primary Main Window

Your application might have multiple main windows, but only one primary main window, which is the starting point for the application. By default, the first main window created in the current project is designated as the primary main window. This designation can be changed in the Application Framework Editor, described in “To Establish Application Framework Behavior” on page 83.

▼ To Edit Properties of a Main Window or a Primary Main Window

Once you have dropped a main window on the desktop do the following to edit its properties. See Appendix B, “Revolving Property Editor,” for descriptions of each of the elements of the property editor.

1. Double-click the main window to display the Revolving Property Editor.

   Or, Choose Properties from the Editors menu of the App Builder primary window, choose Main Window from the Object Type menu, and select the main window in the Objects list.

   The main window will be selected in the Revolving Property Editor.

2. Change the Object Name, if necessary.

3. Change the Window Title to something appropriate.

   This is the label that appears in the title bar of the main window.

4. Type the names of an Icon File, an Icon Mask File, and an Icon Label, if you want an icon to represent the window when it is minimized.

   The Icon File and Icon Mask File must be `xpm` or `xbm` graphics files.

5. Change the User Resize Mode, if appropriate.

   This determines if a user can resize the window in the compiled application.

6. Select Menubar, Toolbar, and Footer, as appropriate, to add these functional areas to the window.

   If you select Menubar, you will want to create menus after you finish editing main window properties. See "Creating and Editing Menus" on page 43 for instructions.

   If you select Toolbar or Footer, you will want to edit the properties of the control panes that comprise these objects after you finish editing main window properties. You can drop controls on the control panes, make connections to programmatic actions, and do other things that can be done to any control pane.

7. Change the Size Policy and Size, as appropriate.

   You will probably want to leave the Size Policy as Fixed while you are creating the application, and change it to Fit Contents as you finish the application, for internationalization and other purposes. When Size Policy is Fit Contents, the window will change size to accommodate changes in the size of objects as the text in the objects changes—or if the font size changes, for example.

8. Set Initial State to Iconic if you want the application to appear as an icon when it is started.

9. If you want the main window to be invisible when the application is started, deselect the visible setting for Initial State.

   If the Visible check box is checked, click it to deselect it.

10. Set Background and Foreground colors, if appropriate.

    Type in a color name if you know it or press mouse button 1 on the Background or Foreground menu button and choose Color Chooser to display the Color Chooser. Select a color and click OK. Background sets the color of the blank pane area of the window. Foreground does nothing that is visible in the completed interface.

11. Click Help Text to add on item help, if appropriate.

    See "To Create Help" on page 66 for instructions.

12. Click OK to apply the changes and dismiss the Revolving Property Editor.

    Unless you are creating a primary main window, you are finished with this task.

13. If you just created a primary main window, choose Application Framework Editor from the Editors menu.
14. Type a Vendor Name and Version number in the Application section of the Application Framework Editor, if appropriate.

    See “To Establish Application Framework Behavior” on page 83 for details about the editor.
15. Click OK in the Application Framework Editor to apply the changes and close the editor.

Creating and Editing Panes, Menus, and Messages 5

Most App Builder objects are dragged from the windows, panes, or controls palettes. Some objects (layered panes, paned windows, menus, and messages) are *created* objects. This chapter explains how to create, use, and edit these objects.

Creating and Editing Pane Entities

There are four types of pane objects on the Panes palette of the App Builder primary window: control pane, text pane, draw area pane, and term pane. In addition, there are three types of *created* pane entities: child panes, layered panes, and paned windows.

Child Panes

A *child pane* is a text pane, term pane, or draw area pane that has been dropped on a control pane and made a “child” of the control pane. In App Builder, for example, the Label field in the Label Property Editor is a text pane that is a child of a control pane.

▼ To Create a Child Pane

1. Drop a text pane, draw area pane, or term pane on a control pane in the interface.

   A message dialog box will be displayed, asking if you want to create the dropped pane as a child of the control pane or as a layered pane.

   Click Cancel if you do not want to create a child pane or a layered pane.

2. Click Child.

   The pane will be instantiated at the drop location, just as if it were a control object. The pane will be a sibling of the control objects on the control pane. You will be able to select the pane and move it around on the control pane just like any other control object.

Layered Panes

A *layered pane* is a “stack” of two or more panes, one on top of the other. In App Builder, for example, the Revolving Property Editor, which is used to display the properties of each of the interface objects, is a layered pane.

▼ To Create a Layered Pane

1. Drop a pane on another pane in the interface.

   A message dialog box will be displayed, giving you the option to create a layered pane.

   If you have dropped a text pane, draw area pane, or term pane on a control pane, you will also have the option to create the object as a child of the control pane.

   Click Cancel if you do not want to create a child pane or a layered pane.

2. Click Layer.

   The pane will be instantiated as a layered pane on top of the pane it was dropped on. Because it is the same size as the original pane, it will obscure the original pane completely. If you resize one of the layers of a layered pane, all panes are resized. See “To View Layered Panes” for instructions for viewing the layers of a layered pane.

   Once you have completed the interface you may want to change the Size Policy of any panes in a layered pane to Fit Contents (the default value is Fixed, which should be retained until the interface is complete). Each of the panes might be a different size.

Note – To unmake a layered pane, select one of the layers and choose Delete or Cut from the Edit menu of the App Builder primary window or from one of the pop-up menus (displayed by pressing mouse button 3 in the interface or in the Module Browser.

▼ To View Layered Panes

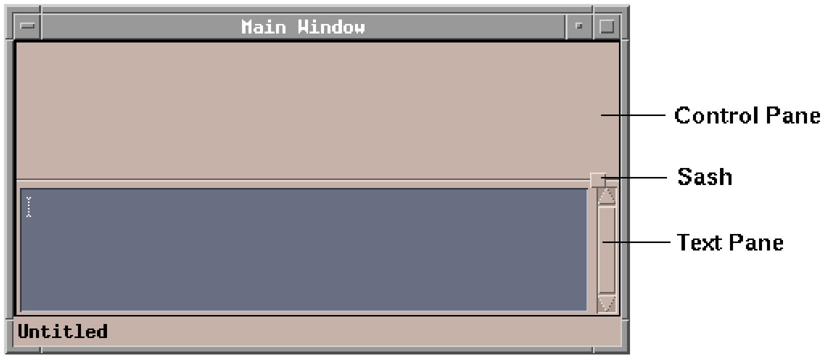
Only one layer of a layered pane is visible. To view other layered panes:

1. Select the visible pane of the layered panes in the interface or in the Module Browser.
2. Choose NextLayer from the interface pop-up menu (displayed by pressing mouse button 3).

   The layer immediately beneath the current pane will be displayed. Repeat this step to view other layers.

Paned Windows

A *paned window* is a combination of two or more panes (control, text, draw area, or term panes, in any combination) into one virtual window with multiple panes, one above the other, separated by a movable sash. While the paned window maintains a constant height, the individual panes become smaller or larger as you move the sash between them.



A paned window's initial size and position are determined by the position and size of its panes: the left margin of the paned window is determined by the left (West) edge of the pane that is furthest to the left. The width of the paned window is determined by the width of its widest pane.

You can set limits on the minimum and maximum heights of any of the panes by setting Pane Height in the Paned Window property editor. See Appendix B, "Revolving Property Editor," for more details.

Once you have created a paned window you can resize the panes by pressing mouse button 1 or mouse button 2 on the sash and moving it up or down.

▼ To Create a Paned Window

1. Drag a pane from the Panes palette and drop it on a main window or a custom dialog.

   If you want the paned window to span the top of the parent window, drop the pane on the top-left corner of the parent. The pane will be attached to the window at its left and top margins with an offset of 0.

2. Resize the pane, if necessary.

   If you want the paned window to span its parent window, drag the right edge of the pane beyond the right edge of the window. The pane will also be attached to this edge.

3. Drag one or more additional panes to the main window or dialog and drop them on an unoccupied portion of the window.

4. Select all panes that you want to be part of the paned window.

   Use mouse button 1 to select one pane and mouse button 2 to select additional panes.

5. Choose Make Paned Window from the Layout menu or from the interface or Module Browser pop-up menu (displayed by pressing mouse button 3).

   The paned window will be created.

   If one of the panes is attached to the right (East) edge of its parent and one or more of the other panes are not attached to the right edge of the parent, a message dialog box will be displayed, explaining that the children of the paned window have different East attachments. Click OK.

▼ To Add a Pane to a Paned Window

1. Drop a pane on the paned window.

   A message dialog box will be displayed, giving you the option to include the new pane as a child of the control pane (if you drop a text pane, draw area pane, or a term pane on a control pane), create as a layered pane, or to add it to the paned window.

2. Click Pane to add the pane to the paned window.

   The new pane will be added to the bottom of the paned window.

▼ To Unmake a Paned Window

1. Select the paned window.

   Select a paned window by clicking at its edge. Be sure you select the paned window and not one of its panes. You will know you have selected the paned window if a dark box is drawn around the paned window and if grab handles appear at the four corners and four midpoints of the paned window.

   Or, open the Module Browser and select the paned window there.

2. Choose Unmake Paned Window from the Layout menu or from the pop-up menu in the interface or the Module Browser (displayed by pressing mouse button 3).

   The panes that made up the paned window will become separate panes again.

Creating and Editing Menus

A *menu* is a list of items with meaningful labels. Each item is connected to a function which is performed when the menu is displayed and the item is selected. This section explains how to create and edit menus, how to attach menus to objects, and how to connect menu items to programmatic functions.

Menus can be attached to menu buttons, menubar items, list items, and any of the four types of panes. A menu is automatically attached to an option menu, so there is no need to attach a menu to it.

Menu Property Editor

The Menu Property Editor is used to create menus. A menu, unlike most of the objects edited in the Revolving Property Editor, is a created object and is not available from the object palettes.

Only properties unique to a menu object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Items, Label, and Item State (Active).

Add New Menu	Adds a new menu to the list of menus.
Edit	Performs edit functions (Cut, Copy, Paste, Delete) on the selected item in the list of menu objects. Cut and Copy place the selected item in a buffer, ready for Paste. Delete removes the item, but does not place it in a buffer.
Tearoff	Specifies whether tearoff is Enabled or Disabled. If tearoff is enabled the selected menu will be "postable." That is, the menu will be displayed until you explicitly dismiss it if you click on the Tearoff indicator (a dotted line).
Item Label Type	Specifies the type of label (String, Graphic, or Separator) for the item selected in the Items list. If Graphic is chosen, Label becomes Graphic Filename. If Separator is chosen, Label or Graphic Filename becomes inactive and Line Style becomes active. A Separator menu item is used to create a visual division in a menu, such as that seen in the Editors menu of the App Builder primary window.

Item Mnemonic	Specifies one of the letters in the selected item as a keyboard shortcut for choosing the item when the menu is posted. The letter specified will be underlined. Pressing the mnemonic letter when the menu is posted causes that item to be chosen. Note that case is significant and that a particular letter can be used as a mnemonic only once within a menu.
[Item] Accelerator	Specifies a keyboard shortcut for choosing the selected item. An accelerator is comprised of a prefix (Ctrl, Alt, Meta, or Shift), an ampersand (&), and a letter (uppercase or lowercase). To make Control-X an accelerator, for instance, type the following: `Ctrl&x` When you display the menu in test mode or in the compiled application, Ctrl+x will be included to the right of the menu item label. If you press the Control key and type `x` with the cursor in the window that contains the menu, the specified action will be performed. You can combine the Shift key with one of the other keys to form a compound prefix, if you wish. To make Shift Control-X an accelerator, type the following: `Shift Ctrl&x`
Line Style	Specifies the type of line style for the selected separator item; active only when Item Label Type is Separator. Choices are None, Etched In, Etched Out, Etched In Dash, Etched Out Dash, Single Line, Double Line, Single Dashed Line, and Double Dashed Line. A separator of the chosen line style will be displayed in the menu instead of a graphic or text label.
Item SubMenu	A menu button and a text field for attaching, de-attaching, creating, or editing a submenu for the selected item in the Items list. If a submenu is attached to the selected item, the name of the submenu will be displayed in the text field.

▼ To Create a Menu

This description assumes you are creating a menu and attaching it to an interface object as two separate procedures. To combine these procedures, see "To Create and Attach a Menu" on page 49.

Note – Menus are available within modules only. Be sure the menu created is in the same module as the object you wish to attach the menu to. Menus are created in the *current module*, which is determined by what is selected in the interface. The Editing Module field in the object information area of the App Builder primary window indicates the current module.

---

1. Display the Menu Property Editor by choosing Menus from the Editors menu in the App Builder primary window.

Note – Choosing Menus from the Editors menu in the App Builder primary window is the same as clicking the Tear-off button in the Revolving Property Editor when the Object Type is Menu.

   Or, display the Revolving Property Editor and choose Menu as the Object Type.

   See "Menu Property Editor" on page 43 for a description of the editor.

   If no menus exist in the current project, the Menu Objects list will be blank and only the Add New Menu and Edit buttons will be active.

   If any menus exist in the current project, they will be listed in the Menu Objects list. One of the menus will be selected in the list and the menu's properties will be displayed for editing.

2. Click Add New Menu.

   A menu will be created with a default Object Name ("menu," "menu2," and so on, depending on how many menus there are in the current module), and with two items in the Items list ("Item1" and "Item2"). The menu will be added to the end of the Menu Objects list, with the name of the current module preceding the menu name. The menu will have default values for Object Name, Tearoff, Items, Item Label Type, Label, and Item State.

If you know you are going to need a number of menus, you could create them all at the same time by clicking Add New Menu the appropriate number of times. You can also create menus that will be used as submenus, to be attached to menu items, at this time.

3. Edit the menu, as described in "To Edit a Menu" on page 46.

   You can edit the menu immediately after creating it or you can edit it later.

   After you have created and edited a menu you will want to attach it to an interface object and make the menu functional by creating connections between menu items and specific actions. See "To Attach an Existing Menu to an Object" on page 48 and "Connecting Menu Items to Actions" on page 72 for instructions. See "To Attach an Existing Submenu to a Menu Item" on page 50 if you want to attach a submenu to a menu item.

▼ To Edit a Menu

After creating a menu you will need to edit the menu: adding menu items, giving the menu items meaningful names, adding submenus, and so on.

1. Display the Menu Property Editor by choosing Menus from the Editors menu in the App Builder primary window.

   Or, display the Revolving Property Editor and choose Menu as the Object Type.

   See "Menu Property Editor" on page 43 for a description of the editor.

1. Change Object Name, if necessary.

   The automatically-generated Object names, which are unique within modules, do not usually need to be changed.

2. Click Enabled to enable the Tearoff function, if necessary.

   This will make the menu "postable," meaning that if you click on the Tearoff indicator (a dotted line) the menu will not be dismissed as soon as you select a menu item. The menu will remain posted until you dismiss it.

3. Add menu items to the Items list, if necessary.

   Click Add Item to add an item after the selected item; choose from the Edit menu button to perform other edit functions.

4. Change Item Label Type for menu items in the Items list, if necessary.

   Choices are String (text), Graphic, or Separator. Label becomes Graphic Filename if Graphic is chosen; Line Style becomes active if Separator is chosen.

5. Type a different Label or Graphic Filename for the selected item, if necessary.

Note – The easiest way to edit labels for menu items is to select the first one in the Items list, thus selecting it in the Label field. Type a new name and click Return. The new name will be displayed in the Items list and the next item in the list will be selected. Continue down the list with this select, type, Return process until all labels are completed.

   If Graphic Item Label Type was chosen, the Graphic Filename must be an `xpm` or `xbm` graphic file.

6. Type an Item Mnemonic, if necessary.

   Type one of the letters in the item label. That letter will be underlined in the menu item label. If the menu is posted, pressing that key will cause the action connected with the menu item to be performed.

Note – Case is significant for mnemonics. The same mnemonic cannot be used more than once in a menu.

---

7. Type an [Item] Accelerator, if necessary.

   An accelerator is comprised of a prefix (Ctrl, Alt, Meta, or Shift), an ampersand (&), and a letter (uppercase or lowercase). See “Menu Property Editor” on page 43 for more information.

8. Choose a Line Style, if Item Label Type is Separator.

   See “Menu Property Editor” on page 43 for the list of choices.

9. Attach an Item Submenu, if appropriate.

   See “To Attach an Existing Submenu to a Menu Item” on page 50 for instructions.

10. Change Item State, if necessary.

    By default the item state is Active. If you want the menu item to be inactive when the application is started, click the Active check box to deselect it.

11. Select Background and Foreground Colors, if necessary.

    Type in a color or choose Color Chooser from the menu and select a color from the Color Chooser. See “To Select Colors from the Color Chooser” on page 32 for details.

12. Click Connections to add programmatic connections to menu items, as necessary.

    See “Making Connections Between Objects” on page 70 for instructions.

13. Click Apply or OK to apply the changes.

    If you click Apply the property editor will remain displayed.

▼ To Attach an Existing Menu to an Object

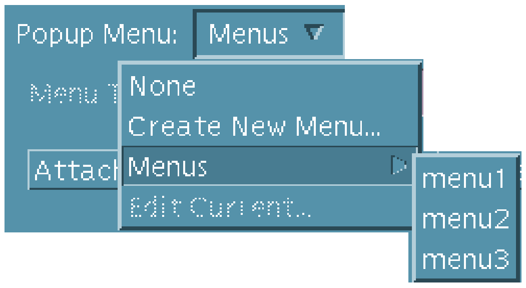
The following instructions assume you have created one or more menus as described in “To Create a Menu” on page 45 and that you are ready to attach a menu to an object in the interface. Menus can be attached to menu buttons, menubar items, list items, and any of the four types of panes. A menu is automatically attached to an option menu, so there is no need to attach a menu to it.

1. Display the Revolving Property Editor with the object to which you wish to attach a menu selected in the editor.

   Double-click the object in the interface or the Module Browser or choose the appropriate Object Type in the Revolving Property Editor and select the desired object in the Objects list.

2. Select a menu to attach to the selected object.

   Press mouse button 1 or 3 on the Popup Menu or Pulldown Menu menu button. Choose the appropriate menu from the Menus submenu.



The name of the selected menu will be displayed in the text field of the Popup Menu or Pulldown Menu.

3. Click OK or Apply.

   The menu will be attached to the selected object. See “Making Connections Between Objects” on page 70 for instructions for making the menu functional.

Note – If you attach a menu to one of the pane objects, the menu will be a pop-up menu, displayed in test mode or in the compiled application by pressing mouse button 3 with the cursor on the pane.

▼ To Create and Attach a Menu

One method of creating and attaching a menu to an object is described in “To Create a Menu” on page 45 and “To Attach an Existing Menu to an Object” on page 48. With the method described here you create and attach the menu at one time. Use whichever method is most convenient.

1. Display the Revolving Property Editor with the object to which you wish to attach a menu selected in the editor.

   Double-click the object in the interface or the Module Browser or choose the appropriate Object Type in the Revolving Property Editor and select the desired object in the Objects list.

2. Choose Create New Menu from the Pulldown Menu or Popup Menu button available for some objects.

Pulldown menus are available for menu buttons and menu bars. Popup menus are available for all pane objects and for list items. An Item SubMenu is available for menus themselves.

The Menu Property Editor will be displayed, with a newly-created menu selected in the Menu Objects list. The menu will have default values for Object Name, Tearoff, Items, Item Label Type, Label, and Item State.

The Object Name will be of the form “object_type_menu,” “object_type_menu2,” and so on, depending on what type of object was selected in the property editor when Create New Menu was chosen and how many menus have been created for the current module. The menu will be added to the end of the Menu Objects list, with the name of the current module preceding the menu name.

3. Edit the menu and click OK to apply the changes and dismiss the Menu Property Editor.

   See “To Edit a Menu” on page 46 for instructions. You can edit the menu later if you like.

4. Click Apply or OK in the Revolving Property Editor to attach the menu to the selected object. See “Making Connections Between Objects” on page 70 for instructions for making the menu functional.

▼ To Attach an Existing Submenu to a Menu Item

The following instructions assume you have created two or more menus as described in “To Create a Menu” on page 45 and that you are ready to attach one of them as a submenu to a menu item.

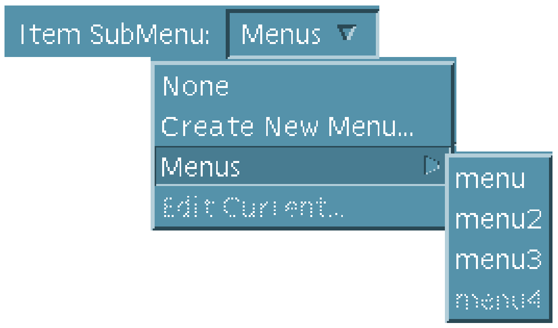
1. Display the Menu Property Editor or the Revolving Property Editor with Menu chosen as the Object Type.

2. In the Menu Objects or Objects list select the menu that contains the menu item to which you want to attach a submenu.

3. In the Items list select the menu item to which you want to attach a submenu.

4. Select a menu to attach to the selected menu item.

   Press mouse button 1 or 3 on the Item SubMenu menu button. Choose the appropriate menu from the Menus submenu.



The name of the selected menu will be displayed in the text field of the Item SubMenu.

5. Click OK or Apply.

   The submenu will be attached to the selected menu item. See “Making Connections Between Objects” on page 70 for instructions for making the submenu functional.

▼ To Create and Attach a Submenu

The following instructions assume you have created one or more menus and that you want to create and attach a submenu to one of the items in one of the menus. With this method you create the submenu and attach it as part of a single procedure. Another method for accomplishing this task is to create the menu as described in “To Create a Menu” on page 45 and to attach it to a menu item as described in “To Attach an Existing Submenu to a Menu Item” on page 50. Use whichever method is most convenient.

Note – When you create and attach a submenu you will be using two editors—one to create the menu and the other to attach the submenu to the menu item. If you start this procedure in the Menu Property Editor, you will be attaching the submenu in the Menu Property Editor but creating it in the Revolving Property Editor. If you start the procedure in the Revolving Property Editor, you will be attaching the menu there but creating it in the Menu Property Editor.

The example below assumes you are starting the procedure in the Menu Property Editor.

---

1. Display the Menu Property Editor by choosing Menus from the Editors menu of the App Builder primary window.
2. In the Objects list select the menu that contains the menu item to which you want to attach a submenu.
3. In the Items list select the menu item to which you want to attach a submenu.
4. Choose Create New Menu from the Item SubMenu menu.

   The Revolving Property Editor will be displayed, with the new menu selected in the Objects list.
5. Edit the menu and click OK to apply the editing changes you made and to dismiss the Revolving Property Editor.

   See “To Edit a Menu” on page 46 for instructions. You can edit the menu later if you like.
6. Click Apply in the Menu Property Editor to attach the submenu to the menu item selected in Step 2. See “Making Connections Between Objects” on page 70 for instructions for making the submenu functional.

▼ To Create and Attach a Help Menu

A help menu at the right end of the menu bar in the application primary main window is a common feature of applications. Do the following to create a help menu and attach it to the Help item of a menu bar. These instructions assume you have included a menu bar in the primary main window and that Help is one of the menu bar items.

1. Display the Revolving Property Editor with Menubar selected in the editor.

   Double-click the appropriate menu bar in the interface or the Module Browser or choose Menubar from the Object Type menu in the Revolving Property Editor and select the desired menu bar in the Objects list. This will normally be the menu bar in the primary main window.
2. Select Help in the Items list.

   This is the Help item on the menu bar.

3. Press mouse button 1 or 3 on the Pulldown Menu menu button. Choose Create New Menu from the Menus submenu.

   The name of the new menu will be displayed in the text field of the Pulldown Menu and the Menu Property Editor will be displayed with the new menu loaded.

4. Edit the menu.

   a. If you want a Help menu that looks like the App Builder Help menu, for instance, add four items to the two default items in the Items list. Select each item in turn and type appropriate labels (Overview, Tasks, Reference, On Item, Using Help, and About [*application_name*], for instance).

   b. Add item mnemonics and accelerators, if appropriate.

      See “Menu Property Editor” on page 43 for details.

   c. Make other changes to the menu, if appropriate.

5. Click OK or Apply in the Menu Property Editor.

   The menu is complete. Click OK to apply the changes and dismiss the editor.

6. Click OK or Apply in the Revolving Property Editor.

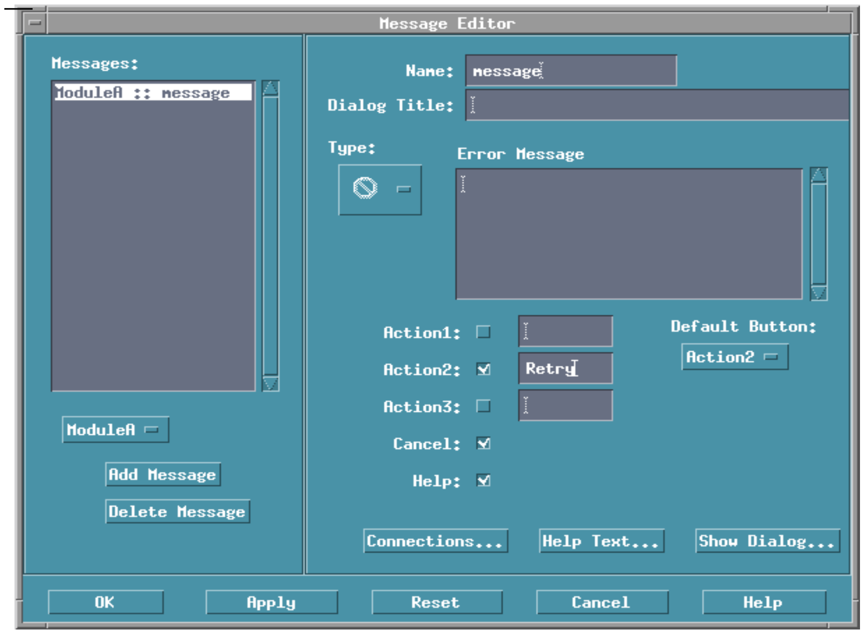
   The Help menu has been attached to the Help item in the menu bar. Click OK to apply the changes and dismiss the editor.

Creating and Editing Messages

This section describes the Message Editor and explains how to create and edit message dialog boxes.

Message Editor

The Message Editor is used to create various types of messages to be displayed at appropriate times in the compiled application. It is shown in Figure 5-1 and then described. See “To Create a Message Dialog Box” on page 56 and “To Edit a Message” on page 57 for instructions on its use.



ModuleA :: message

ModuleA

Name: message

Dialog Title:

Action1: ☐

Action2: ☑ Retry

Action3: ☐

Cancel: ☑

Help: ☑

Default Button: Action2

Figure 5-1 Message Editor

Messages	Lists all messages for the current project. The module name precedes the message name in the list.

Module menu	Specifies the module for which you wish to add a new message. The module name precedes the message name in the Messages list.
Add Message	Adds a new message to the Messages list and to the current project. The message is for the module selected in the module option menu.
Delete Message	Deletes the selected message.
Name	Specifies the instance name of the current message object. Messages are given names such as “message,” “message2,” “message3,” by default.
Dialog Title	Specifies the title that will appear at the top of the message dialog box.
Type	Specifies the type of message to be created. The choices are Error, Information, Working, Question, and Warning. The message type appears above the message text pane. The appropriate message icon appears in the message dialog box in the compiled applications.
Message text pane	A text pane for entering the text of the message. Press Return when you want the text to start a new line. The label above the text pane varies, depending on what type of message you have chosen.
Button check boxes	Specifies which buttons will be included at the bottom of the message dialog box. Each message type has a different set of buttons specified by default; these default choices can be changed. Actions associated with the Action1, Action2, Action3, and Cancel buttons are set in the Connections Editor. See “To Create a Message Dialog Box” on page 56 for detailed instructions.
Default Button menu	Specifies the default button for the selected message dialog.
Connections	Displays the Connections Editor for specifying what functions to call for each of the Action buttons and the Cancel button.
Help Text	Displays the Help Editor, in which you write help text to be displayed when the Help button is clicked in the message dialog box.

Show Dialog	A push button for displaying the selected message in a message dialog box that looks like the actual dialog box in the compiled application. Click one of the buttons other than Help to dismiss the dialog box.

See “Property Editor: Common Buttons” on page 135 for descriptions of the buttons at the bottom of the editor.

▼ To Create a Message Dialog Box

See “Message Editor” on page 54 for descriptions of each of the fields in the editor. See “To Connect a Non-Modal Message to a Function” on page 58 for a discussion of how to connect messages to the functions that cause them to be displayed, with examples.

1. Choose Messages from the Editors menu of the App Builder primary window to display the Message Editor.
2. Choose the module to which you want to add a message in the option menu below the Messages list.
3. Click Add Message.

   A unique name (“message,” “message2,” and so on, depending on how many messages are in the current module) will be displayed in the Name field. The module name and the message name will be added to the Messages list.
4. Modify the Name if you wish.

   This is the name used to identify the message internally—in the Connections Editor, for instance. This name is not displayed in the compiled message dialog box.
5. Type a title for the message dialog in the Dialog Title field.

   This will appear in the title bar of the compiled message dialog box.
6. Choose a message type from the Type menu.

   The icon for the message type will be displayed in the Type menu and the message type (Error, Information, Working, Question, or Warning) will be displayed above the text pane where the message text pane.

7. Type the message text in the message text pane, pressing Return when you want a new line to start in the compiled message.
8. Specify which buttons will appear in the message dialog box by clicking the check boxes below the message text pane and typing the labels you want on the Action1, Action2, and Action3 buttons.

   Each of the message types includes a default set of buttons that you can modify:

   - Error: Action2 (Retry), Cancel, Help.
   - Information: Action1 (OK), Help.
   - Working: Action1 (Close), Action2 (Stop), Help.
   - Question: Action1 (Yes), Action2 (No), Help.
   - Warning: Action2 (Continue), Cancel, Help.
9. Choose a default button from the Default Button menu.

   This is the button that will have an extra border when the message dialog box is displayed. This is the button that will be activated if Return is pressed. Each of the message types has a default Default Button that you can modify:

   - Error: Action2
   - Information: Action1
   - Working: Action1
   - Question: Action1
   - Warning: Action2
10. Click the Help Text button and create help text, as appropriate.

    See "To Create Help" on page 66 for instructions.
11. Click OK or Apply to apply the changes.

    The Message Editor will be dismissed if you click OK.

▼ To Edit a Message

1. Choose Messages from the Editors menu of the App Builder primary window to display the Message Editor.
2. Select the message you want to edit in the Messages list.
3. Edit the message, as appropriate.

- To delete a message, click Delete Message.
- To modify the dialog box title, click in the Dialog Title text field and type the new label.
- To change the message type, choose a different Type icon.
- To modify the message text, click in the message text pane and type the appropriate changes.
- To change the available buttons, select the check boxes and type new button labels, if appropriate.
- To change the default button, choose another from the Default Button menu.
- To modify help text, click Help Text, make the changes in the Help Editor, and click OK in the Help Editor.

4. Click OK or Apply to apply the changes.

   The Message Editor will be dismissed if you click OK.

▼ To Connect a Non-Modal Message to a Function

See “Example: Writing Code for Messages” on page 59 for a discussion of how to connect a message to the function that causes it to be displayed, with examples. In particular, read that section to see how to attach a modal (blocking) message to a function.

1. Display the Connections Editor by clicking Connections in the Message Editor or by choosing Connections from the Editors menu of the App Builder primary window.

   If you select a message in the Message Editor and click Connections, the selected message will be selected in the Source list of the Connections Editor. You can skip the next two steps.

2. Display messages in the Source list by choosing Message from the Source menu.

3. Select a message in the Source list.

4. Choose Call Function as the Action Type.

   This activates the When menu on the Source side of the Connections Editor.

5. Choose a When item (Action1, Action2, Action3, or Cancel Activated, depending on which buttons were checked in the Message Editor).

6. Type the name of the Function to be called when the selected button is selected.

   When code is generated, this function is created in <*module_name*>`_stubs.c`. You will have to substitute appropriate code before running `make`.

7. Click Connect to create the connection.

   The connection will be displayed in the View list at the bottom of the Connections Editor.

8. Repeat the previous three steps for each button except Help.

9. Click Cancel to dismiss the Connections Editor.

Example: Writing Code for Messages

Once you have created a message as described in "To Create a Message Dialog Box" on page 56, you must determine when and how it should be displayed. Usually messages are displayed after a certain piece of logic has been executed. For example, if a user types digits in a text field that is designed to accept a name, you will want to post an error message informing the user that digits are not valid.

Message boxes in Motif can be displayed in one of two ways: *modally* or *non-modally* (equivalently, *blocking* or *non-blocking*). The App Builder code generator (`dtcodegen`) supplies two routines, corresponding to the two modes of display. They are found in `dtb_utils.c` and are named:

- `dtb_show_modal_message()`
- `dtb_show_message()`

If you want to display a particular message modally, use `dtb_show_modal_message()`. If you want to display a particular message non-modally, use `dtb_show_message()`.

One of the key differences in the way these two types of messages are handled is in how the application determines which button was pressed by the user in the message dialog box. For non-modal messages callbacks are added to each button via the Connections Editor. When the user clicks a button the corresponding callback is called. Since modal dialogs are *blocking*, the button callbacks are *not* called. Instead, the value is returned by `dtb_show_modal_message`, which indicates which button is pressed by the user.

▼ To Write Code for Modal Messages

If a message is to be displayed modally, use `dtb_show_modal_message()`. This routine returns a value which indicates which message box button the user has pressed. The value is an enum that is defined in `dtb_utils.h`:

```
/*
 * Returns answer value for modal MessageBox
 */
typedef enum {
      DTB_ANSWER_NONE,
      DTB_ANSWER_ACTION1,
      DTB_ANSWER_ACTION2,
      DTB_ANSWER_ACTION3,
      DTB_ANSWER_CANCEL,
      DTB_ANSWER_HELP
} DTB_MODAL_ANSWER;
```

You can then examine the return value (for example via a switch statement) and execute the appropriate piece of code.

Here's an example of displaying a message modally. Say that you have created a simple application, named `foo`. The project is named `foo.bip` and consists of one module, `foo.bil`. The module `foo.bil` consists of a main window, control pane, and two text fields, one for the user to enter a person's first name and the other to enter the last name. If the user types digits, an error message will be posted, informing the user that digits are not allowed, and giving the user a couple of options. The user can start over, which means the text entered will be erased, or the user can continue, which means that the text entered will be left intact, giving the user discretion as to how to modify the text.

A call-function connection is made for both text fields, which will be called each time the user types something. The function for the first text field will check if the character typed is a digit. If so, it will post the error message modally:

```
void
verify_first_nameCB(
    Widget widget,
    XtPointer clientData,
    XtPointer callData
)
{
/*** DTB_USER_CODE_START vvv Add C variables and code below vvv ***/
    char                *text = (char *)NULL;
    int                 textlen = 0;
    DTB_MODAL_ANSWER    answer = DTB_ANSWER_NONE;
    DtbFooMainwindowInfo instance = (DtbFooMainwindowInfo)
clientData;
/*** DTB_USER_CODE_END   ^^^ Add C variables and code above ^^^ ***/

/*** DTB_USER_CODE_START vvv Add C code below vvv ***/

    text = XmTextFieldGetString(widget);
    if ((text != NULL) && (*text != NULL))
    {
        textlen = strlen(text);
        if (isdigit(text[textlen-1]))
        {
            dtb_foo_message_initialize(&dtb_foo_message);
            answer = dtb_show_modal_message(instance->textfield,
                        &dtb_foo_message, NULL, NULL, NULL);
            switch (answer)
            {
                case DTB_ANSWER_ACTION1:        /* Start Over */
                    XmTextFieldSetString(widget, "");
                    break;

                case DTB_ANSWER_ACTION2:/* Continue */
                    break;
            }
        }
    }

    /*** DTB_USER_CODE_END   ^^^ Add C code above ^^^ ***/
}
```

▼ To Write Code for Non-Modal Messages

If you want to post a non-modal message, use `dtb_show_message()`. Since this function is not modal and does not return a return value, callbacks for the message box buttons should be specified via the Connections Editor, as described in “To Connect a Non-Modal Message to a Function” on page 58. The buttons that are specified for the message box are displayed as When items for the message object in the Connections Editor.

Using the same example as above, make the last name text field display the error message non-modally if the user types a digit. As previously mentioned, first you'll need to make a couple of call-function connections for the two buttons in the message box, labelled "Start Over" and "Continue." When code is generated, add code to those routines to do the right thing. The start over routine will clear out the text field and the continue routine will do nothing, in this case.

```
void
verify_last_nameCB(
    Widget widget,
    XtPointer clientData,
    XtPointer callData
)
{
/*** DTB_USER_CODE_START vvv Add C variables and code below vvv ***/
    char                *text = (char *)NULL;
    int                 textlen = 0;
    DtbFooMainwindowInfo instance = (DtbFooMainwindowInfo)
clientData;

/*** DTB_USER_CODE_END   ^^^ Add C variables and code above ^^^ ***/

/*** DTB_USER_CODE_START vvv Add C code below vvv ***/

    text = XmTextFieldGetString(widget);
    if ((text != NULL) && (*text != NULL))
    {
        textlen = strlen(text);
        if (isdigit(text[textlen-1]))
        {
            dtb_foo_message_initialize(&dtb_foo_message);
            dtb_show_message(instance->textfield,
                        &dtb_foo_message, NULL, NULL);
        }
    }

/*** DTB_USER_CODE_END   ^^^ Add C code above ^^^ ***/
}

void
start_overCB(
    Widget widget,
    XtPointer clientData,
    XtPointer callData
)
{
/*** DTB_USER_CODE_START vvv Add C variables and code below vvv ***/

DtbFooMainwindowInfo instance = (DtbFooMainwindowInfo) clientData;

/*** DTB_USER_CODE_END   ^^^ Add C variables and code above ^^^ ***/

/*** DTB_USER_CODE_START vvv Add C code below vvv ***/

XmTextFieldSetString(dtb_foo_mainwindow.textfield2, "");

/*** DTB_USER_CODE_END   ^^^ Add C code above ^^^ ***/
}


void
continueCB(
    Widget widget,
    XtPointer clientData,
    XtPointer callData
)
{
/*** DTB_USER_CODE_START vvv Add C variables and code below vvv ***/
/*** DTB_USER_CODE_END   ^^^ Add C variables and code above ^^^ ***/

/*** DTB_USER_CODE_START vvv Add C code below vvv ***/
/*** DTB_USER_CODE_END   ^^^ Add C code above ^^^ ***/
}
```

The two routines above, start_overCB() and continueCB(), are added as callbacks for the two buttons via the call to dtb_show_message(). Here is the code fragment that adds the callback (from dtb_utils.c):

```
    /* Add Callbacks if necessary */
if (mbr->action1_callback != (XtCallbackProc) NULL)
    XtAddCallback(msg_dlg, XmNokCallback, mbr->action1_callback,
    NULL);
if (mbr->cancel_callback != (XtCallbackProc) NULL)
    XtAddCallback(msg_dlg, XmNcancelCallback, mbr->cancel_callback,
    NULL);
if (mbr->action2_callback != (XtCallbackProc) NULL)
    {
        action_btn = dtb_MessageBoxGetActionButton(msg_dlg,
DTB_ACTION2_BUTTON);
        if (action_btn != NULL)
            XtAddCallback(action_btn, XmNactivateCallback,
                          mbr->action2_callback, NULL);
    }
    if (mbr->action3_callback != (XtCallbackProc) NULL)
    {
        action_btn = dtb_MessageBoxGetActionButton(msg_dlg,
DTB_ACTION3_BUTTON);        if (action_btn != NULL)
            XtAddCallback(action_btn, XmNactivateCallback,
                          mbr->action3_callback, NULL);
    }
```

The structure `mbr` contains all the necessary information for the message. The structure is filled in with the values specified in the Message Editor when the message object was created via the `dtb_&_&_initialize()` routine—in this example, `dtb_foo_message_initialize()`.

Adding Functionality to the Interface 6

Once you have laid out an interface you may want to add help to interface elements, make programmatic connections between objects, specify drag and drop behavior, and specify application framework behavior (including internationalization, resource file creation, session management, and ToolTalk message handling).

Creating Help and Help Connections

Two kinds of help—object help and a help volume—can be accessed from an App Builder application. Object help is created in App Builder, as explained in “To Create Help” on page 66. A help volume is created separately from App Builder, and is accessed in your compiled application from the Help menu or by clicking More in a help dialog box. See the *Help System Author's and Programmer's Guide*, which is included in the desktop Help Developer's Kit, for instructions for creating a help volume.

About App Builder Help

With App Builder you can create help for any object in the interface—a control, a pane, or a window. Help is created in the Help Editor, as described in "To Create Help" on page 66. In test mode or in the compiled application, help is displayed in a in the following ways:

- Press F1 with the cursor over an interface window.

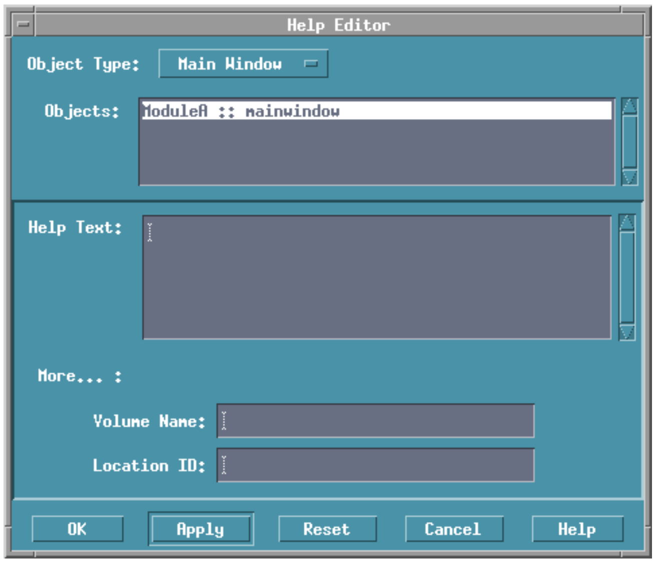
  If help exists for the object with input focus, it is displayed. If there is no help for the object with input focus but help exists for a parent window, help for that window will be displayed.

- Click the Help button in a window or dialog box.
- Choose On Item from the Help menu and click on an object in the interface.

See "To Test On Item Help" on page 99 for instructions for testing On Item help. If help is not available for a particular child object (a control or a pane) but is available for the parent of the child object (a pane or a window), help for the parent object is displayed.

▼ To Create Help

1. Display the Revolving Property Editor.
2. Choose the Object Type for which you want to write help.
3. Select the object for which you want to write help.
4. Click Help Text to display the Help Editor with the appropriate object selected.



5. Type help text in the Help Text pane.

   Press Return when you want a new line to start in the compiled help dialog box.

6. Type a Volume Name if appropriate.

   This is the name of a help volume.

7. Type a Location ID, if appropriate.

   This is the helptag location ID that will provide more information about the selected object.

Note – You must create help for an object if you want access to a help volume from a help dialog box. If you create help for an object and include a Volume Name and Location ID, the More button will be active in the help dialog box.

---

8. Click OK or Apply to apply the changes.

   If you want to add help to other objects, choose the appropriate Object Type in the menu, select the appropriate object, and repeat the previous two steps.

   Click OK to apply the changes and dismiss the Help Editor.

▼ To Connect a Help Menu to On Item Help

One of the standard items in a Help menu is On Item Help, which is used to display help for a specific object in an interface. The instructions below assume you have included a menu bar in a main window and that you have attached a Help menu to the Help item in the menu bar. See “To Create and Attach a Help Menu” on page 52 for instructions.

1. Choose Menus from the Editors menu in the App Builder primary window.

   The Menu Property Editor is displayed.

2. Select the Help menu in the Objects list.

3. Select one of the items in the Items list as the On Item Help item.

4. Type On Item or other appropriate text in the Label text field.

5. Include an item mnemonic, if appropriate.

   An item mnemonic specifies one of the letters in the selected item as a keyboard shortcut for activating the menu item when the menu is posted. The letter specified will be underlined in the menu item. Case is significant for mnemonics.

6. Include an item accelerator, if appropriate.

   An item accelerator specifies a keyboard shortcut for choosing the selected item. An accelerator is comprised of a prefix (Ctrl, Alt, Meta, or Shift) , an ampersand (&), and a letter (upper- or lowercase). To make Control-X an accelerator, for instance, type `Ctrl&x`.

7. Click Apply.

   The changes to the Help menu will be applied.

8. Click Connections to display the Connections Editor.

   The Connections button is at the bottom of the Menu Property Editor.

9. Choose Menu Item in the Source menu.
10. Select the On Item Help item in the Source list.
11. Choose Activate On Item Help from the Action Type menu.
12. Click Connect.

When you choose the On Item Help item in the Help menu in test mode or in the compiled application, the cursor will become an arrow with a question mark. Move the cursor over an object and click mouse button 1 to display On Item help for the selected object (or for one of its parent objects if no help is available for the object itself). See “To Test On Item Help” on page 99 for more information.

▼ To Connect a Help Menu to a Help Volume

After creating a help menu and attaching it to the Help item in a menu bar as explained in “To Create and Attach a Help Menu” on page 52, do the following to connect menu items to specific locations in a help volume. See “To Connect a Help Menu to On Item Help” on page 68 for instructions for connecting the On Item help item in the Help menu to the On Item help function.

1. Display the Connections Editor.

   Click Connections in the Revolving Property Editor or in the Menu Property Editor or choose Connections from the Editors menu.

2. Choose Menu Item from the Source option menu.
3. Select one of the Help menu items from the Source scrolling list.
4. Choose Access Help Volume from the Action Type option menu.
5. Type the name of the help volume in the Volume text field.
6. Type the appropriate location ID in the Location text field.
7. Click Connect to make the connection.

Making Connections Between Objects

In its simplest form a connection is a programmatic relationship between a source object and a target object: when I click on Button A I want Dialog Box B to be displayed. This type of connection is described below in "To Make a Connection between Two Objects" on page 70. Different types of connections from menu items are described in "To Connect a Menu Item to a Predefined Action" on page 72, "To Connect a Menu Item to a Call Function" on page 73, and "To Connect a Menu Item to an Execute Code Action" on page 74.

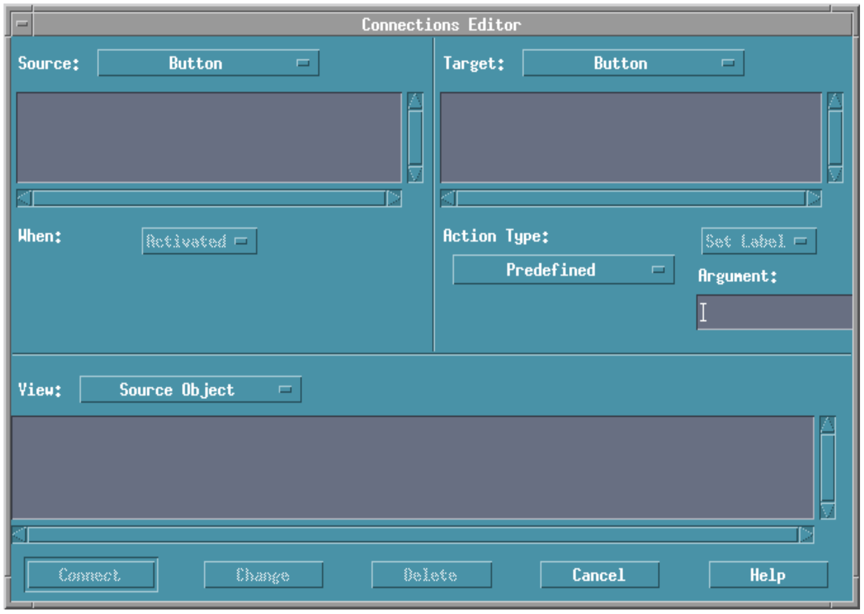
Other types of connections (to On Item Help and to a help volume) were discussed in "To Connect a Help Menu to On Item Help" on page 68 and in "To Connect a Help Menu to a Help Volume" on page 69. In "To Connect a Non-Modal Message to a Function" on page 58, a message dialog box is connected to the function which causes the dialog box to be displayed.

▼ To Make a Connection between Two Objects

1. Select the source and target objects.
   - **By "drag-linking"**: While holding down the Control key, position the mouse cursor over the intended source object, press mouse button 1, drag the cursor to the intended target object, and release the mouse button. This can be done in the interface or in the Module Browser (or between the interface and the Module Browser).

     A line with a "plug" at its end will extend from the source as you move the mouse. The target object will be highlighted with a dark box. The Connections Editor will be displayed, with the source and target objects selected.

   - **Through the Connections Editor**: Display the Connections Editor by choosing Connections in the Editors menu. Choose the object type you want as the source object in the Source menu, and select the object you want as the source in the Source list. Then choose the object type you want as the target object in the Target menu, and select the object you want as the target in the Target list.



2. Choose an action in the When menu.

   This is the action on the source object that will cause an action to be performed on the target object. Choices vary, depending on the source object type.

3. Choose an action to be performed on the target in the Action Type menu.

   Different target action types require different subsequent action by you:

   - Predefined: Choose an action from a second option menu.
   - Call Function: Type the name of a function in the Function text field. You will also have to write code for the call function, as described in “Adding User Code to Generated Code” on page 106.
   - Execute Code: Type the code to be performed in the Execute Code Editor and click OK in the editor.

4. Click Connect or press Return to make the connection.

   The connection will be displayed in the View list at the bottom of the Connections Editor.

Connecting Menu Items to Actions

Once you have created a menu and attached it to an object as described in “To Create a Menu” on page 45 and “To Attach an Existing Menu to an Object” on page 48 you need to connect a meaningful action to each item in each menu. Choices for target actions are Predefined, Call Function, Execute Code, Activate On-Item Help, and Access Help Volume.

Connecting menu items to the first three types of actions are described below. See “To Connect a Help Menu to On Item Help” on page 68 and “To Connect a Help Menu to a Help Volume” on page 69 for instructions for making help connections.

▼ To Connect a Menu Item to a Predefined Action

Only Predefined target actions are described in this section. See “To Connect a Menu Item to a Call Function” on page 73 and “To Connect a Menu Item to an Execute Code Action” on page 74 for information about those connections.

1. Display the Connections Editor.

   Click Connections at the bottom of the Revolving Property Editor or choose Connections from the Editors menu of the App Builder primary window.

2. Choose Menu Item from the Source menu.

   All of the menu items in the current project will be listed.

3. Select a menu item from the list below the Source menu.

   This is the item from which the connection will be made.

4. Choose Predefined as the target action type from the Action Type menu.

   The Target menu will be activated.

5. Choose the appropriate type of object from the Target menu.

   This is the type of object that will be acted on when the When action is performed on the source menu item.

6. Select an object in the list of Target items.

   This is the specific object that will be acted on when the When action is performed on the source menu item.

7. Choose a When action for the Source menu item.

   Choices are Activated, Created, and Destroyed.

8. Choose a target action from the option menu to the right of Action Type.

   The choices vary depending on the target type.

9. Click Connect to make the connection.

   The connection will be displayed in the View list at the bottom of the Connections Editor.

The designated target action will be performed in the compiled application when the When action is performed on the menu item.

Depending on the source When and target action, you may be able to test the connection in Test mode. See "To Test Menus in a Module" on page 100 for instructions.

▼ To Connect a Menu Item to a Call Function

Only the Call Function target action is described in this section. See "To Connect a Menu Item to a Predefined Action" on page 72 and "To Connect a Menu Item to an Execute Code Action" on page 74 for information about those connections.

1. Display the Connections Editor.

   Click Connections at the bottom of the Revolving Property Editor or choose Connections from the Editors menu of the App Builder primary window.

2. Choose Menu Item from the Source menu.

   All of the menu items in the current project will be listed.

3. Select a menu item from the list below the Source menu.

   This is the item from which the connection will be made.

4. Choose Call Function as the target action type from the Action Type menu.

   The Function text field will be activated.

5. Type the name of the function to be called in the Call Function text field.

   This is the function that will be called when the When action is performed on the source menu item. See “Adding User Code to Generated Code” on page 106 for information about incorporating user code into the generated code.

6. Choose a When action for the Source menu item.

   Choices are Activated, Created, and Destroyed.

7. Click Connect to make the connection.

   The connection will be displayed in the View list at the bottom of the Connections Editor.

▼ To Connect a Menu Item to an Execute Code Action

Only the Execute Code target action is described in this section. See “To Connect a Menu Item to a Predefined Action” on page 72 and “To Connect a Menu Item to a Call Function” on page 73 for information about those connections.

1. Display the Connections Editor.

   Click Connections at the bottom of the Revolving Property Editor or choose Connections from the Editors menu of the App Builder primary window.

2. Choose Menu Item from the Source menu.

   All of the menu items in the current project will be listed.

3. Select a menu item from the list below the Source menu.

   This is the item from which the connection will be made.

4. Choose Execute Code as the target action type from the Action Type menu.

   The Execute Code Editor will be displayed.

5. Choose a When action for the Source menu item.

   Choices are Activated, Created, and Destroyed.

6. Type the code to be executed in the Execute Code Editor.

   The Execute Code Editor will be displayed. Type the code in the editor. See "Adding User Code to Generated Code" on page 106 for information about incorporating user code into the generated code.

7. Click OK in the Execute Code Editor to apply the changes and dismiss the editor.

8. Click Connect in the Code Generator window to make the connection.

   The connection will be displayed in the View list at the bottom of the Connections Editor.

The code will be executed in the compiled application when the When action is performed on the menu item.

▼ To Edit an Existing Connection

Once you have created a connection you can modify the connection, delete it, or create an additional connection by selecting an existing connection, modifying it, and saving it as a new connection.

1. Choose Connections from the Editors menu in the App Builder primary window.

   The Connections Editor will be displayed.

2. Choose the source object type whose connection you want to view from the View menu at the bottom of the Connections Editor.

   If you want to edit a connection with a button as a source object, for instance, choose Button from the View menu. All connections in the current project with button as source object will be displayed in the View list.

   If you want to view all connections for a particular source object, choose Source Object in the View menu and select the object in the Source menu. All connections for the selected object will be displayed.

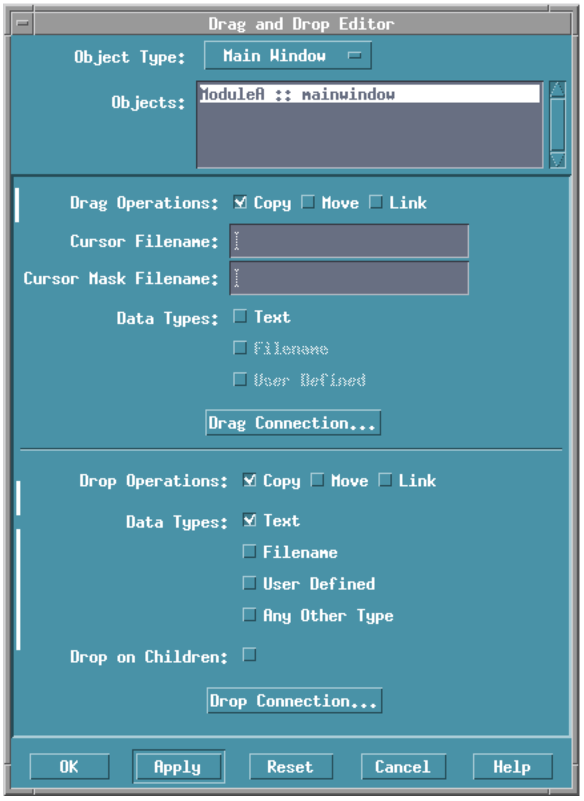
3. Select the connection you want to edit in the View list.

   The source and target objects will be selected in the Source and Target lists at the top of the editor. Their When and Action Type choices will be displayed.

4. Edit the connection.
   - To delete the selected connection, click Delete.
   - To modify the selected connection, make changes to any of the choices (source object, When action, target object, Action Type) and click Change.
   - To add a connection similar to the selected connection, modify any of the choices and click Connect. A new connection will be created.
5. Click Cancel to dismiss the Connections Editor.

Drag and Drop Editor

Use the Drag and Drop Editor to establish drag and drop behavior for interface objects.



Object Type: Main Window

Objects: ModuleA :: mainwindow

Drag Operations: [✓ Copy] Move Link

Cursor Filename:

Cursor Mask Filename:

Data Types: Text Filename User Defined

Drop Operations: [✓ Copy] Move Link

Data Types: [✓ Text] Filename User Defined Any Other Type

Drop on Children:

Object Type	An option menu for choosing the type of object (Control Pane, Custom Dialog, Draw Area Pane, Label, or Main Window) for which you wish to establish drag and drop behavior.
Objects	A scrolling list for selecting a specific object for which you wish to establish drag and drop behavior.
Drag Operations	Check boxes for specifying which types of operations (Copy, Move, Link) will be legal for the selected object.
Cursor Filename	A text field for typing the name of the graphics file that contains the graphical representation of the cursor that will be displayed as a drag from the selected object is being performed.
Cursor Mask Filename	A text field for typing the name of the graphics file that contains the bitmap which determines the shape of the visible representation of the cursor beneath the cursor mask. The cursor mask acts like a stencil, allowing only the pixels in the cursor that correspond to pixels in the mask to be visible.
Data Types	Check boxes for specifying Text, Filename, and User Defined as legal data types for drag operations.
Drag Connection	A push button to display the Connections Editor for creating the Call Function connection that makes the dragged-from operation functional.
Drop Operations	Check boxes for specifying which types of operations (Copy, Move, Link) will be legal for the selected object.
Data Types	Check boxes for specifying Text, Filename, User Defined, and Any Other Type as legal data types for drop operations.
Drop on Children	A check box for specifying whether a child of the selected object will be a legal drop site; this is relevant only if the child object is specified as a legal drop site.
Drop Connection	A push button to display the Connections Editor for creating the Call Function connection that makes the dropped-on operation functional.

▼ To Establish Drag and Drop Behavior

1. Choose Drag and Drop from the Editors menu of the App Builder primary window.

   The Drag and Drop Editor is displayed.

2. Choose an Object Type.
3. Select an object in the Objects list.
4. Select the Drag Operations you want to be legal for the selected object.
5. To display a special cursor when a drag operation is being performed from the selected object, type the names of graphics files in the Cursor Filename and Cursor Mask Filename fields.
6. Select the Data Types that will be legal for drag operations.
7. Click Drag Connection.

   The Connections Editor is displayed.

8. Choose Dragged From as the When action in the Connections Editor.
9. Choose Call Function as the Action Type in the Connections Editor.
10. Type a name for the called function in the Function text field in the Connections Editor.

    This is the name of the function that will be called when a drag operation is performed. You will have to edit the `stubs.c` file to make the called function do something useful. See “Adding User Code to Generated Code” on page 106 for information.

11. Click Connect in the Connections Editor.
12. Select which Drop Operations will be legal.
13. Select the Data Types that will be legal for drop operations.
14. Check Drop on Children if you want a drop operation on a child of the selected object to be legal.

    This is relevant only if the selected object has a child which is designated as a legal drop site.

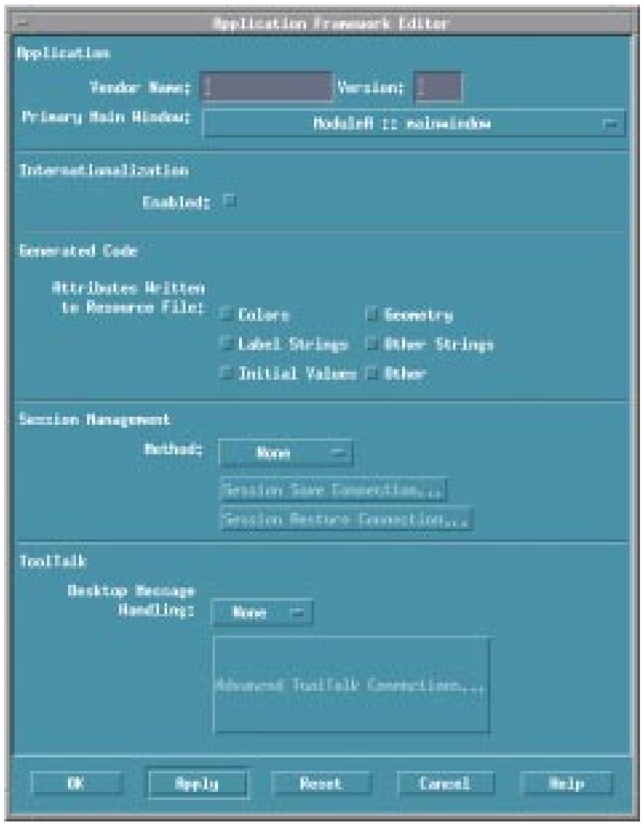
15. Click Drop Connection to display the Connections Editor.

16. Choose Dropped On as the When action in the Connections Editor.
17. Choose Call Function as the Action Type in the Connections Editor.
18. Type a name for the called function in the Function text field in the Connections Editor.

    This is the name of the function that will be called when a drop operation is performed. You will have to edit the `stubs.c` file to make the called function do something useful. See "Adding User Code to Generated Code" on page 106 for information.
19. Click Connect in the Connections Editor.
20. Click OK or Apply in the Drag and Drop Editor to apply the changes.

Application Framework Editor

Use the Application Framework Editor to specify basic functionality in the application for internationalization, resource file attributes, session management, and ToolTalk message handling.



Application Vendor Name	A text field for typing an optional string, which will be stored in the source code. Used in the call to initialize ToolTalk (if ToolTalk is enabled).
Application Version	A text field for typing an optional string, which will be stored in the source code. Used in the call to initialize ToolTalk.
Application Primary Main Window	An option menu for specifying the primary main window of the application being developed. An application may have more than one main window, but only one primary window. This window is typically the window which is first displayed when the

application is opened. By default the first main window dropped on the desktop in a new project is the primary window.

Internationalization Enabled	A check box for specifying whether internationalization is enabled; if checked, turns on XPG4-compliant internationalization in the generated code for the project. In the [module]`_ui.c` file, all labels and strings for objects are generated, enclosed by the `catgets(3C)` call, which is used to fetch the appropriate localized version of the string at runtime. If internationalization is turned on, `dtcodegen` will also automatically generate and maintain the message catalog ([project]`.msg`) which maps to the generated `catgets(3C)` calls.
Generated Code	Check boxes for specifying which categories of object attributes (which map to Xt Resources) should be written into a Resource file instead of placing them directly in the [module]`_ui.c` file -- which is the default. Any attribute (resource) which is specified in a Resource file -- and not directly in the code -- can be modified without recompiling the application. The Attribute categories are as follows: Colors: Background, Foreground Label Strings: Label String, Title Initial Values: Initial Value Geometry: X, Y, Width, Height, all attachment attributes Other Strings Other
Session Management Method	An option menu for specifying the method of session management (None, Command Line, Session File, or Both), and two push buttons (Session Save Connection, Session Restore Connection) for displaying the Connections Editor and making appropriate connections.
ToolTalk Desktop	

Message Handling	An option menu for specifying what level of the ToolTalk Desktop Message Alliance protocol the application will participate in, and a push button (Advanced ToolTalk Connections) for displaying the Connections Editor. The ToolTalk desktop protocol is a set of predefined ToolTalk messages which communicate desktop-type events or requests to a running application. App Builder support for ToolTalk is provided at three levels: None, Basic, or Advanced, as described below.

None: There is no participation in the ToolTalk Desktop Protocol; no ToolTalk code is generated.

Basic: The ToolTalk library responds to Desktop messages in categories 1-3 in a predefined and standard way. Code is generated in `main()` which initializes ToolTalk and calls the function which tells ToolTalk to handles these messages. At this level, you do not need to write any special application code.

Advanced: The ToolTalk library responds to messages in categories 1 and 2, but the application is notified (via callback) when messages in categories 3 & 4 are received.

If you choose Advanced, you must use the Connections Editor to identify which messages the application wishes to handle. If you click the Advanced ToolTalk Connections button, the Connections Editor will be displayed with Application as the Source object type. The When option menu lists four ToolTalk choices: ToolTalk Do Command, ToolTalk Get Status, ToolTalk Pause/Resume, and ToolTalk Quit. The only valid action a ToolTalk connection is Call Function; your callback function will be called when the ToolTalk message is received.

At this level code is generated in [*project*]`.c:main()` which initializes ToolTalk and sets up the Desktop Protocol so that the callbacks defined in the Connections Editor will be called when the corresponding message is received. Each user-defined callback contains descriptive comments describing what the application is expected to do in response to the message. These callbacks are also generated in [*project*]`.c`.

▼ To Establish Application Framework Behavior

1. Choose Application Framework from the Editors menu in the App Builder primary window to display the editor.
2. Type a Vendor Name and Version number in the text fields in the Application section, if appropriate.

   These are used in the call to initialize ToolTalk, if ToolTalk is enabled.
3. Choose a different primary main window, if appropriate.
4. Set Internationalization to Enabled, if appropriate.

   Internationalization generates labels and strings for objects with a call that fetches the appropriate localized version of the string at run time. It also generates and maintains a similar message catalog.
5. Select the attributes you want to be written to the Resource file in the Generated Code section.

   The categories you select are written to a resource file instead of directly to the module file; these attributes, therefore, can be modified without recompiling the application.
6. Choose a Method (None, Command Line, Session File, or Both) in the Session Management section, as appropriate.
7. Select Session Save Connection and/or Session Restore Connection, as appropriate, to make connections in the Connections Editor.
8. Choose a Desktop Message Handling level (None, Basic, or Advanced) in the ToolTalk section, as appropriate.

   See "Application Framework Editor" on page 79 for more about ToolTalk message handling.
9. If you did not choose Advanced in the previous step, click OK to apply the changes made and dismiss the Application Framework Editor.
10. Click Advanced ToolTalk Connections if you chose Advanced in the previous step.
11. Choose the appropriate ToolTalk function from the When menu in the Connections Editor.
12. Choose Call Function as the Action Type.

13. Type in the name of the appropriate call function.

    This is the name of the function that will be called when a ToolTalk operation is performed. You will have to edit the `stubs.c` file to make the called function do something useful. See “Adding User Code to Generated Code” on page 106 for information.

14. Click Connect to make the connection.

15. Click OK in the Application Framework Editor to apply the changes and dismiss the editor.

Grouping and Attaching Objects 7

This chapter discusses how to group and attach objects for dynamic layout behavior.

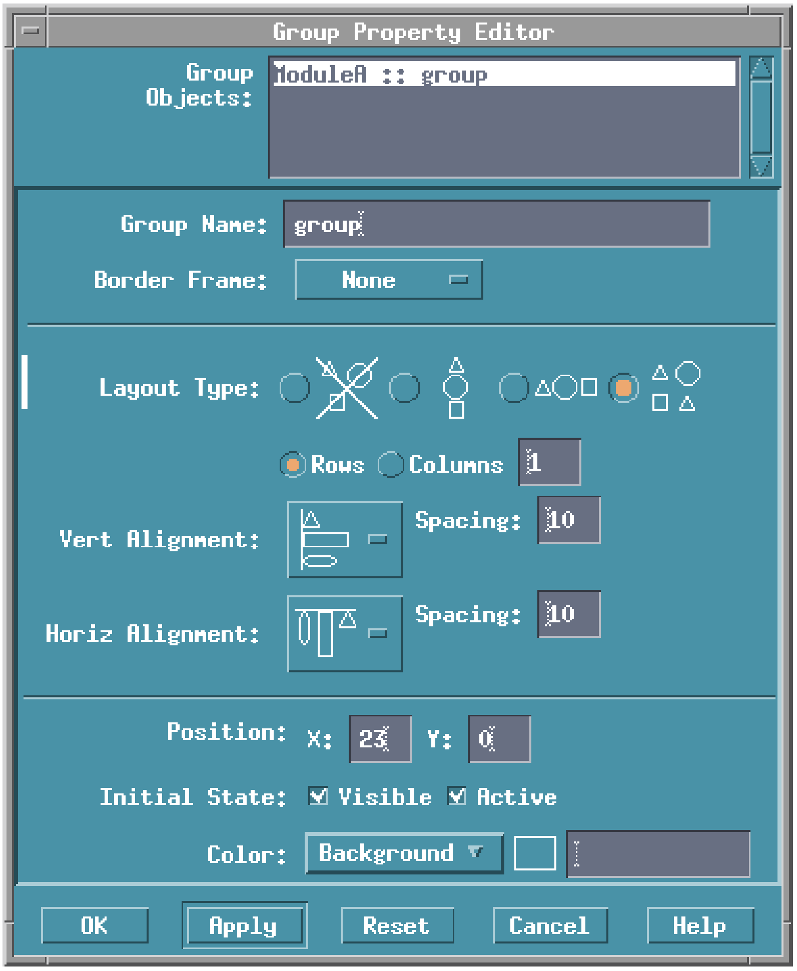
Grouping Objects

In order to ensure that interface objects maintain consistent spacing and size relationships, regardless of text changes (including internationalization changes) and resizing of windows, you may need to group control objects and to attach objects to each other.

A *group* is a collection of objects that can be treated as a unit. Once the objects in a group are positioned as desired, the group can be moved, maintaining the relative positioning of the individual objects. Because groups use dynamic layout for positioning objects, spacing and alignment in the group are maintained if any of the objects in the group change size.

Group Property Editor

See “Group Property Editor” on page 140 for a description of the editor and each of its elements.



▼ To Create a Group

1. Select the control objects you want to be part of the group.

   You can select the objects either in the interface or the Module Browser, and you can select the objects in whatever manner is most convenient. See “To Select Control Objects” on page 25 for instructions.

2. Choose Group from the Layout menu or the pop-up menu (displayed by pressing mouse button 3 with the cursor in the window interface or in the Module Browser).

   A rectangular box will be drawn around the group in the interface, indicating that the group is selected. Note that Ungroup is active in the Layout and pop-up menus; this is only true when a group is selected.

   A new object will be displayed and selected in the Module Browser—an object called "group" (or "group2," and so on, if other groups exist in the module). The group object is the parent of the objects that comprise the group. Group members cannot be moved independently. Any attempt to move an object in a group will cause the entire group to move.

▼ To Edit Group Properties

Group properties, including horizontal or vertical alignment and spacing between objects, are set in the Group Property Editor.

1. Double-click the group in the interface or in the Module Browser to display the Group Property Editor.

   In the interface you will have to click in the space between group members to select the group. The group will be selected in the Group Property Editor.

   You can also display the Group Property Editor by choosing Groups from the Editors menu.

2. Select the group to be edited from the Group Objects list, if necessary.

3. Type a new name for the group, if necessary.

4. Choose a border frame style if you want the group to have a border in the completed interface (no border is the default).

   Border frame style choices are shadow out, shadow in, etched out, etched in, and none.

5. Select a Layout Type.

   Choices are as-is, vertical, horizontal, and row-column.

   Depending on what you select, either the Vert Alignment or Horiz Alignment option menu, or both, will be active. If you select rows-columns, the Rows and Columns radio buttons will be active, also.

6. Designate the number of Rows or Columns (if row-column layout was selected).

   The number of columns will be determined automatically if you designate the number of rows, and the number of rows will be determined automatically if you designate the number of columns.

7. Choose a vertical alignment (if either vertical alignment or row-column layout type was chosen).

   The choices are align on left edge of objects (the default), align on colons/labels, align on middle of objects, or align on right edge of objects.

8. Designate vertical spacing (if either vertical alignment or row-column layout type was chosen).

   The absolute values are in pixels; 10 is the default.

9. Choose a horizontal alignment (if either horizontal alignment or row-column layout type was chosen).

   The choices are align on top edge of objects (the default), align on middle of objects, or align on bottom edge of objects.

10. Designate horizontal spacing (if either horizontal alignment or row-column layout type was chosen).

    The absolute values are in pixels; 10 is the default.

11. Deselect Visible if you do not want the objects in the group to be visible when the application is opened.

12. Deselect Active if you do not want the objects in the group to be active when the application is opened.

13. Click OK or Apply to apply the changes.

    The Group Property Editor will be dismissed if you click OK.

▼ To Ungroup Objects in an Interface

1. Select the group in the Module Browser or in the interface.

   In the interface, click between objects in a group to select the group. You will know the group is selected if a box appears around two or more objects.

   If you can't select a group in the interface or if you want to be sure to select the right group in an interface with many groups, open the Module Browser. Groups are shown in the Module Browser by name of group; if you select the group in the Module Browser, it is also selected in the interface.

2. Choose Ungroup from the interface pop-up menu (displayed by pressing mouse button 3 in an interface window).

   The objects are no longer part of the group. You can now select any of the objects and move it independently of the other objects.

▼ To Create a Border around an Object

The group function can be used to create a border around an individual object, such as a label.

1. Select the object in the interface.
2. Choose Group from the Layout menu or the interface pop-up menu.

   The object will be part of a group.

3. Display the Group Property Editor.
4. Select the group you want to put a border around.

   If you double-click the group in the Module Browser, the Group Property Editor will be displayed, with the group selected.

5. Choose the Border Frame style you want to add to the object.
6. Click OK to apply the change and dismiss the Group Property Editor.

Attaching Objects

Attachments (and groups, which are based on attachments) establish dynamic layout behavior for objects in the interface. Dynamic layout behavior ensures that objects will maintain consistent relationships during resize activities. Attachments enable an internationalized application to work well in a number of locales.

All child objects are attached by their top and left edges to the top and left edge of their parent object, by default. Thus a control pane dropped on a main window is attached by its left and top edges to the left and top edges of the main window. Similarly, a button dropped on the control pane is attached to the control pane.

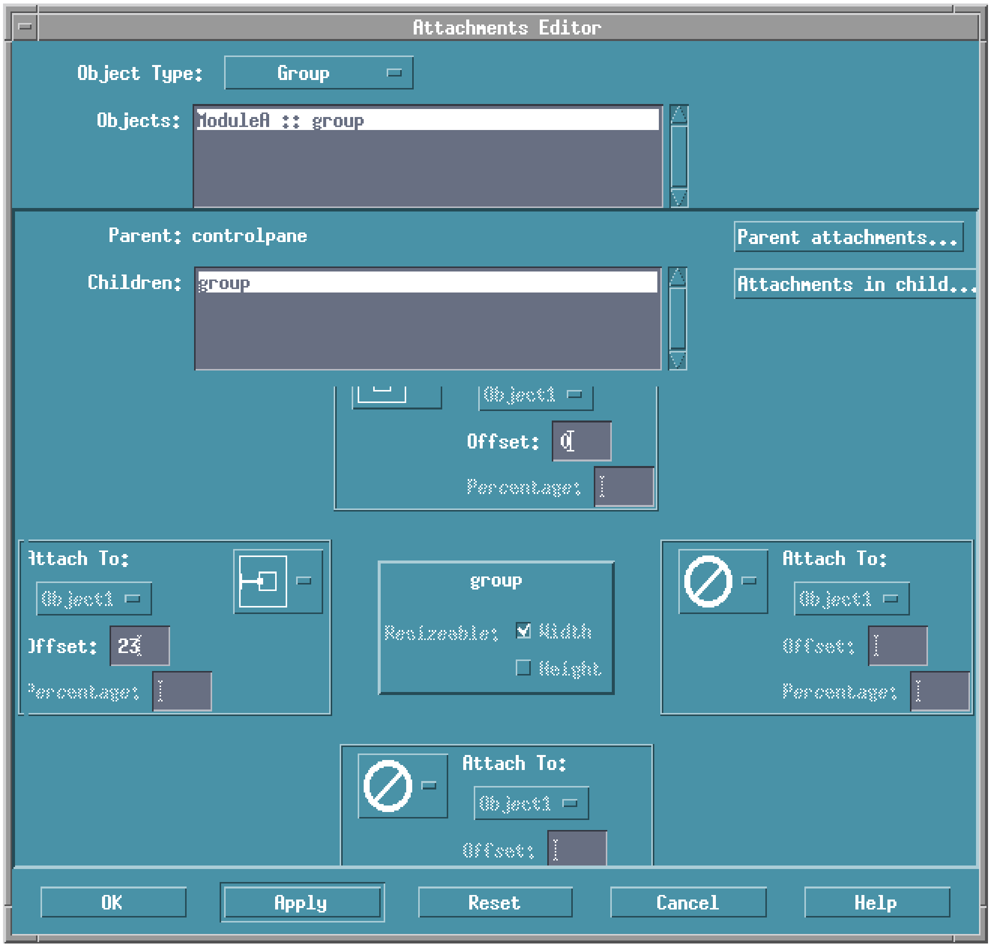
If the parent object is resized in an upward or leftward direction, the child object moves with the parent, maintaining the distance from the top and left edge of the parent.

If a pane object is dropped on the top or left edge of its parent it will be attached to that edge with an offset of 0. If it is dropped some distance to the right and below the left and top edges of its parent, it will have positive offsets.

If a pane object is resized from its right and bottom edges so that it spans its parent object, it will be attached to the right and bottom edges of its parent.

Attachments Editor

Used to attach objects to each other for layout purposes, the Attachments Editor is described below.



Object Type	An option menu for choosing the type of object for which you want to make attachments. Some object types (custom dialog, file selection dialog, main window) do not have parents and are not included in the menu.
Objects	A scrolling list for selecting the object for which you want to make attachments.
Parent	A text field that indicates the parent of the selected object.
Children	A scrolling list that lists the children of the Parent object.
Parent attachments Attachments in child	Radio buttons for displaying the attachments of the parent of the selected object or the attachments of the children of the selected object. Some objects (draw area pane, term pane, text pane) cannot have children and may be children of a main window or custom dialog; thus, neither Parent attachments nor Attachments in child will be active. If the pane is a child of another pane, though, or if it is part of a layered pane, Parent attachments will be active.
Attach To	Option menus for choosing the type of attachment for the selected object and what to attach the object to, and text fields for specifying the Offset (in pixels) from the selected object and its parent or sibling and for specifying the Percentage offset of the selected object from its parent. The option menu below "Attach To:" is for choosing which sibling to attach to and is active only for sibling attachments (two small squares). The Offset text field is active for absolute (pixel) attachments only; the Percentage text field is active for percentage attachments only. The selected object is shown in the center of its four possible attachments. The attachments, starting at the top and going clockwise, are top edge of selected object, right edge of selected object, bottom edge of selected object, and left edge of selected object. Each of the possible types of attachments is described below; by default an object is attached at its top and left edges

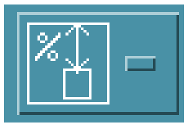
to the top and left edges of its parent. The selected object (the object at the center of the four Attach To boxes) is the controlling object: if you move this controlling object, the pixel or percentage offset is changed; click Reset to see current values after moving an attached object.

An ascending line from the top edge of a small square to the top edge of its surrounding box represents an absolute (pixel offset) attachment of the top edge of the selected object to the top edge of its parent. If the parent object is resized, the selected object will retain its pixel offset from the top edge of its parent. The offset will change if the selected object is moved.

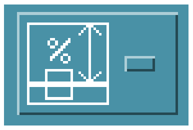
A descending line from the top edge of a small square to the bottom edge of its surrounding box represents an absolute (pixel offset) attachment of the top edge of the selected object to the bottom edge of its parent. The offset will change if the selected object is moved. This value will be negative, since x values are positive as they ascend and negative as they descend.

Two vertically-aligned squares connected by a vertical line represents an absolute (pixel offset) attachment of the top edge of the selected object to the bottom edge of its sibling (a sibling is another object with the same parent). The offset will change if the selected object is moved. This icon is inactive if the selected object has no siblings. This value will be negative if the top edge of the selected object is above the bottom edge of its sibling.

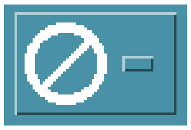
Two horizontally-aligned squares connected by a horizontal line to the centers of their top edges represents an absolute (pixel offset) attachment of the vertical center of the selected object to the vertical center of its sibling (a sibling is another object with the same parent). The offset will change if the selected object is moved. This icon is inactive if the selected object has no siblings. This value will be negative if the center of the selected object is above the center of its sibling.



A square with a two-headed arrow and a percentage sign above it represents a percentage offset attachment of the top edge of the selected object to the top edge of its parent. The offset will change if the selected object is moved. If the parent object is resized, the selected object will retain its percentage offset from the top edge of the parent.



A square with a percentage sign above it and a two-headed arrow between the center line of the square and the top of the surrounding box represents a percentage offset attachment of the center of the selected object to the top edge of its parent. The offset will change if the selected object is moved. If the parent object is resized, the selected object will retain its percentage offset from the top edge of the parent.



A circle with a diagonal line through it represents no attachment from the edge (top, left, bottom, or right) to another object. By default a dropped object has no right or bottom edge attachments.

See “Property Editor: Common Buttons” on page 135 for descriptions of the buttons at the bottom of the editor.

Note – Descriptions of the attachments to the right, bottom, and left edges of the selected object are correlatives of the descriptions of the top-edge attachments above. Substitute "bottom" for "top" and "top" for "bottom" for bottom-edge attachments. Substitute "right" for "top" and "left" for "bottom" for right-edge attachments. Substitute "left" for "top" and "right" for "bottom" for left-edge attachments.

Attachment Example: Custom Dialog

See an App Builder custom dialog object for an example of attachments. Each of the buttons at the bottom of the custom dialog are attached to the top and sides of their enclosing dialog panel. They are attached five pixels from the top of the panel and varying percentages from the left edge of the panel (Button1 left edge 10%, right edge 30%; Button2 40% and 60%; Button3 70% and 90%).

The left edge of Button1 will always be 10% from the edge of the panel and the right edge of Button1 will always be 30% from the edge of the panel. Button1 will therefore always be as wide as 20% of the total width of the panel. Button2's edges are 40% and 60% from the left edge of the panel; Button3's edges are 70% and 90% from the left edge of the panel.

The three buttons will grow and shrink as the panel grows and shrinks, and the distance between them will always be 10% of the total width of the panel.

▼ To Attach Objects in an Interface

See "Attachments Editor" on page 91 for an illustration of the editor and descriptions of its elements.

1. Choose Attachments from the Editors menu in the App Builder primary window to display the Attachments Editor.

   The Attachments Editor can also be displayed by clicking the Attachments button in a property editor or by choosing Attachments from the interface or Module Browser pop-up menu.

2. Choose the object type you want to attach to its parent or siblings.

3. Select the object that you want to attach.

4. Select an attachment type.

   If you choose an icon with one small square you are making an attachment from a child object to its parent. If you choose an icon with two small squares you are making a sibling attachment. See "Attachments Editor" on page 91 for descriptions of the types of attachments.

   When you make an attachment, the selected object—the object in the center of the four Attach To boxes—is the controlling object. That is, this object can be moved without affecting its parent or sibling. The offset value or percentage value will change to reflect the changed relationship between the two objects.

   On the other hand, if you move the other object—the object to which the selected object is attached—the selected object will move so as to maintain its relationship with the other object.

   You may have to click Reset after moving an object in the interface before the change is noted in the Attachments Editor.

5. Click OK or Apply to apply the changes.

   If you click OK, the Attachments Editor will be dismissed.

Testing Menus, Help, and Connections 8

Many functions of your interface can be tested without generating code and making the application. In both Test Shown Modules and Test Project mode all build windows except the App Builder primary window are closed, and the App Builder primary window is inactive except for the Build button and the Help menu.

If your project is small, you will probably want to test the entire project. If it is large, you may want to test only selected modules, thus saving the time it takes to load a large project. In Test Project mode the entire project is available. Windows that are designated as not visible at startup (as are custom dialogs by default, for instance) will not be visible.

See “To Show a Hidden Module” on page 18 for instructions if you are going to use Test Shown Modules.

▼ To Test a Project or Selected Modules

1. Click Test Project or Test Shown Modules in the App Builder primary window.

   Depending on which button you selected, all modules in the current project or only shown modules will be tested.

2. Test help, if appropriate.

   See “To Test On Item Help” on page 99 for instructions.

3. Test menu displays, if appropriate.

   See “To Test Menus in a Module” on page 100 for instructions.

4. Test connections, if appropriate.

   See “To Test Connections in a Project” on page 101 for instructions.

5. Click Build to return to build mode.

▼ To Test Help Volume Access

These instructions assume you have created a help menu and attached it to a Help menu on the menu bar of a main window, as described in “Creating Help and Help Connections” on page 65.

1. Display the modules to be tested, if necessary.

   If you are not going to test the entire project, you will need to show the modules to be tested. See “To Show a Hidden Module” on page 18 for instructions.

2. Click Test Shown Modules or Test Project, as appropriate.

   Click Test Project to test the entire project. Click Test Shown Modules to test selected modules.

3. Test help volume access by choosing one of the help volume chapters (Overview, Tasks, Reference, for example) from the Help menu.

   A help volume window with the appropriate help text will be displayed, if the help viewer (`dthelpview`) is accessible and the proper connection has been made to the compiled help volume. See “Creating Help and Help Connections” on page 65 for instructions for creating help and making connections to it. Dismiss the help window when you are finished with it.

4. Click Build to return to build mode.

▼ To Test On Item Help

These instructions assume you have created a help menu and attached it to a Help menu on the menu bar of a main window, as described in “Creating Help and Help Connections” on page 65.

1. Display the modules to be tested, if necessary.

   If you are not going to test the entire project, you will need to show the modules to be tested. See “To Show a Hidden Module” on page 18 for instructions.

2. Click Test Shown Modules or Test Project, as appropriate.

   Click Test Project to test the entire project. Click Test Shown Modules to test selected modules.

3. Test help volume access by choosing one of the help volume chapters (Overview, Tasks, Reference, for example) from the Help menu.

   A help volume window with the appropriate help text will be displayed, if the help viewer (`dthelpview`) is accessible and the proper connection has been made to the compiled help volume. See “Creating Help and Help Connections” on page 65 for instructions for creating help and making connections to it. Dismiss the help window when you are finished with it.

4. Test On Item help by choosing On Item from the Help menu.

   The cursor will turn into an arrow and a question mark.

5. Move the cursor over an interface object and click.

   If the object (or one of its parent objects) has help text, it will be displayed in a quick help window.

6. Click the More button in the quick-help window, if it is active.

   The help volume will be displayed, at the location specified in the Location ID for the selected object in the Help Editor. Dismiss the help window when you are finished with it.

7. Click the Close button in the quick help window to dismiss it.
8. Click Build to return to build mode.

▼ To Test Menus in a Module

In Test Shown Modules mode all windows in the currently-shown modules will be displayed, including those whose initial state is not set Visible. See “To Test a Project or Selected Modules” on page 98 if you want to test the entire project, with not-Visible windows hidden.

1. Display the module to be tested, if necessary.

   See “To Show a Hidden Module” on page 18 for instructions.

2. Click Test Shown Modules.
3. Click or press on the items in a menu bar, if appropriate.

   The menus will be displayed. If you select a menu item that is connected to certain predefined functions (Show or Hide a dialog, Access Help Volume, Activate On Item Help, for example), the function will be performed.

4. Click each button menu, as appropriate.

   The menus will be displayed. If you select a menu item that is connected to certain predefined functions (Show or Hide a dialog, Access Help Volume, Activate On Item Help, for example), the function will be performed.

5. Press mouse button 3 on a pane or list item to display a pop-up menu, if appropriate.

   The menus will be displayed. If you select a menu item that is connected to certain predefined functions (Show or Hide a dialog, Access Help Volume, Activate On Item Help, for example), the function will be performed.

6. Click Build to return to build mode.

▼ To Test Connections in a Project

1. Display the modules to be tested, if necessary.

   See "To Show a Hidden Module" on page 18 for instructions.

2. Click Test Project.

   All build windows except the App Builder primary window will be closed, and the primary window will be inactive except for the Build button and the Help menu. Only windows in the project with an initial state set to Visible will be displayed.

3. Click a button or choose a menu item that has a testable connection.

   The following connections should work in test mode as they will work in the compiled application:

   - Show
   - Hide
   - Set Value
   - Set Text
   - Access Help Volume
   - Activate On Item Help
   - Enable
   - Disable

   If you connect a button to a custom dialog, for instance, specifying the button as the source object, Activated as the When action, the custom dialog as the target object, and Show as the Action Type, the custom dialog will be displayed when you click the button.

   Connections to Call Function and Execute Code will be noted by messages to standard out.

   Connections to Application Framework, ToolTalk, and message dialogs are not supported in test mode.

Generating Code and Building an Application

9

This chapter describes the Code Generator and its use to generate code, add user code to generated code, make the application, and run the compiled application. See “Code Generator Window” on page 125 for an illustration of the Code Generator window and descriptions of its elements.

Making and Running an Application

Two scenarios are described below: in the first, you build and run an application in one step. In the second case, you generate code, compile the code, and run the application in separate steps.

In either case, if you have made changes to the project that have not been saved, a message dialog box will be displayed, telling you that you have unsaved edits and giving you the choice of cancelling the generate code process or saving the project. If you choose to save the project, you will have to specify where to save the project if it has not been saved before.

▼ To Make and Run in One Step

1. Choose Code Generator from the File menu of the App Builder primary window.

   The Code Generator is displayed. .

2. Click Make & Run to generate code, build the application, and run it.

   If you have saved the project and all goes well, a number of messages will be displayed in the output pane at the top of the Code Generator. The final message will be "Running: ./[projectname]" and the application will run.

   At the least, the application primary window will be displayed. Any windows whose visibility is not set to yes at application startup will be hidden. Depending on what functionality you included that does not require user code, the application might do a variety of things. Menus can be displayed, some connections can be tested, On Item help can be displayed, and so on.

Note – Ultimately, you must write some code to complete the application. For example, any Call Function callbacks specified in the Connections Editor will have to be substituted for. See “Adding User Code to Generated Code” on page 106 for more information.

▼ To Generate Code, Make, and Run Separately

1. Click Generate Code to generate code for the current project.

   As the code generator runs, messages are displayed in the output pane at the top of the Code Generator window. The final message should be "Completed successfully." A number of files will be created, including Makefiles, project files, module files, and two `dtb_utils` files. You can look at the files in the term pane at the bottom of the Code Generator window.

2. Click Make to build the application.

   More messages will be displayed in the Output Pane as the application is compiled. The final message again should be "Completed successfully." A few more files will be created, including object files and the executable application file, which has the name you gave the project.

3. Click Run to run the application.

   The application will be started—as if you had typed the name of the executable at the command line.

4. Click Abort to quit the application.

   This will terminate the application, closing all windows. You can also click Abort to terminate code generation or `make` operations started in the Code Generator window.

▼ To Set Code Generator Options

To change the options that determine what code is generated and other Code Generator functions:

1. Choose Code Generator from the File menu of the App Builder primary window to display the Code Generator window.

2. Choose Generator from the Options menu to display the Code Generator Options dialog box.

3. Select one of the Generate Code For options (Entire Project, Main Only, Specific Modules Only, Specific Modules and Main).

   If you select Specific Modules or Specific Modules and Main, the list of modules is active. Select the names of the modules you want to generate code for in the list.

4. Click Don't Merge if you do not want your hand-edited code merged with the generated code.

Note – Do not select Don’t Merge unless you are sure you want to destroy the user code.

---

5. Choose a different message reporting option if you wish.

   Choices are Report Normal Messages, Be Silent, and Be Verbose.

6. Type Make Arguments, if appropriate.

   These arguments will be included when you click Make or Make & Run.

7. Type Run Time Arguments, if appropriate.

   These arguments will be included when you click Run or Make & Run.

8. Click Reset to Defaults to set all fields to their default values.

   Default values are Generate Code For Entire Project, Merge user code with generated code, and Report Normal Messages.

9. Click OK or Apply to make the changes.

   The Options dialog box will be dismissed if you click OK.

▼ To Set Environment Options

1. Choose Code Generator from the File menu of the App Builder primary window to display the Code Generator window.

2. Choose Environment from the Options menu to display the Environment Options dialog box.

3. Type a variable in the Variable Name text field.

   You might want to change PATH, for instance.

4. Click Get to display the current value for the variable in Variable Name.

   The value of the variable will be displayed in the Value pane.

5. Make a change to Value and click Set to change the value of the variable.

   This change is made for this App Builder session only.

6. Click Reset to reset Value to its value outside this session of App Builder.

7. Click Cancel to dismiss the dialog box.

Adding User Code to Generated Code

When you generate code for the interface you have developed by clicking Generate Code in the Code Generator window or running `dtcodegen` from the command line, a number of files are generated in the project folder. If your project is called “test” and it has one module, called “mod1,” for instance, the following files will be created:

- `Makefile` (plus Makefiles for other platforms)

- `dtb_utils.c`
- `dtb_utils.h`
- `mod1.bil` (module file)
- `mod1_stubs.c`
- `mod1_ui.c`
- `mod1_ui.h`
- `test.bip` (project file)
- `test.c`
- `test.h`
- `Test` (resource file)

If you have made Call Function or Execute Code connections in the Connections Editor, those connections will show up in the generated code. All of the areas of the generated code that my be modified by you are marked with comments of the form:

```
/* DTB_USER_CODE START */
/* DTB_USER_CODE_END */
```

The area between the `START` and `END` comments are considered a "user segment." Any text (even non-C code) may be added within a user segment, and the code generator will preserve this code in all future versions of the code. Each user segment begins with a comment that suggests what type of code should be added in that segment, or what state the application is in when that segment is executed. These suggestions are purely informational, and may be ignored.

Neither App Builder nor the code generator verify that the code added by you is legal C code. It is your responsibility to ensure that any file you modify can be processed satisfactorily by your compiler.

If you wish to destroy all of the hand-edited code, you must either explicitly select Don't Merge from the Options dialog of the Code Generator Window, or run `dtcodegen` with the `-nomerge` option. This should be done only with great caution, as large amounts of work may be lost.

Under no circumstances should the generated comments be modified. If they are modified, code generation will fail, and the resulting file will very likely be uncompilable. A backup file, with the extension `.BAK`, is preserved in the current directory to help recover from such mistakes.

The user code segments appear in strategic places in the code, to allow you a great deal of freedom in customizing the generated application. All code related to `main()` and application-wide data and structures are defined in <*projectname*>`.h` and <*projectname*>`.c`. In these files, fields may be added to the Xt resource data structure for the application, new developer-defined data types and variables may be added, and the application's startup procedures may be amended.

Each <*modulename*>`_stubs.c` file contains user segments for modifying the effects of generated connections. Your code may be added both before and after the automatically-generated code is executed.

In addition, each file contains a user segment at the top of each file that can be used to add a custom header or copyright notice.

▼ To Generate Code from the Command Line

To generate App Builder code from the command line, run `dtcodegen`. Usage is described below.

Usage: `dtcodegen` [*options*] [*project-file*] [*module-file* [*module-file*] ...]

Code is generated for each module specified on the command line, or for all modules in the project, if no modules are specified. If no project file is specified, a project file containing the specified module(s) is searched for in the current directory.

Files with extension `.bip` are assumed to be BIL project files, files with `.bix` extension are assumed to be encapsulated BIL files, and files with a `.bil` extension are assumed to be BIL module files.

Options (* = default, + = default with no project file)

`-help (-h)`	Print out this help message.
`-main`	Write file containing main().
`-changed`	Only generate files that have changed.
`* -merge`	Merge generated _stubs.c files with previous version.
`-nomerge`	Don't merge existing and new stubs file.
`* -project (-p)`	Specify a project to generate code for.

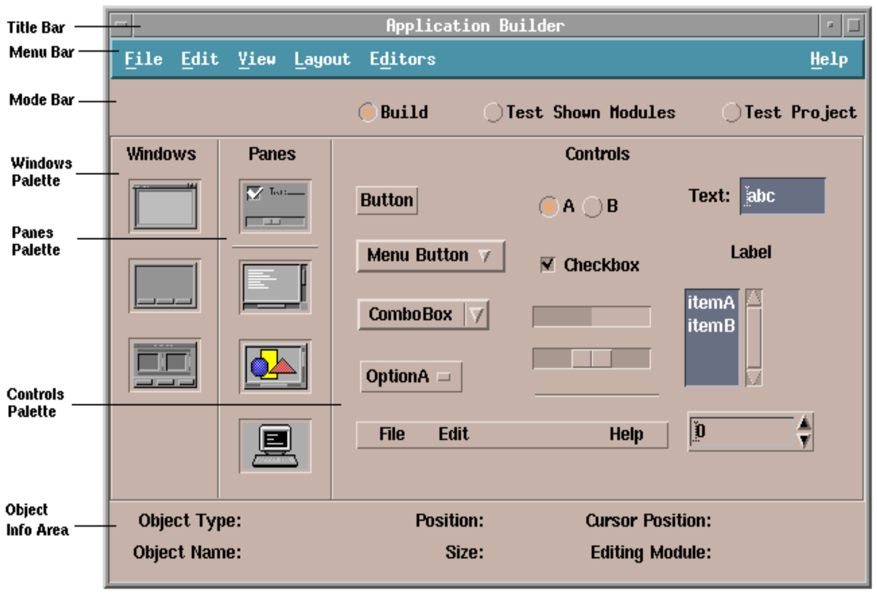
`-noproject (-np)`	Use default project settings, ignore project file.
`+ -showall`	Application shows (maps) all windows at startup.
`* -noshowall`	Application shows (maps) only initially-visible windows.
`-silent (-s)`	Silent mode, no messages written.
`-verbose (-v)`	Verbose mode, detailed progress messages.

App Builder Windows and Dialog Boxes

A

This appendix describes the major windows and dialog boxes in App Builder, including illustrations of the windows and dialog boxes and descriptions of the window and dialog box elements.

App Builder Primary Window



The App Builder primary window is the starting point for building a graphical user interface. The interface is created by dragging objects from the App Builder object palettes (Windows, Panes, and Controls) to the desktop, editing the properties of the resultant interface objects, and adjusting the layout of the interface. See “Overview of the App Builder Process” in Chapter 1, “Getting Started,” for a summary of the steps involved in creating an interface.

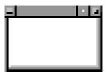
Title bar	Includes the name of the application, “Application Builder,” the window manager menu, a minimize button, a maximize button, the name of the current project (if one is open), and a "(Save Needed)" indication if the current project has changed since being saved.
Mode bar	Includes Build, Test Shown Modules, and Test Project radio buttons for specifying build and test modes. Build is for designing and building an interface.

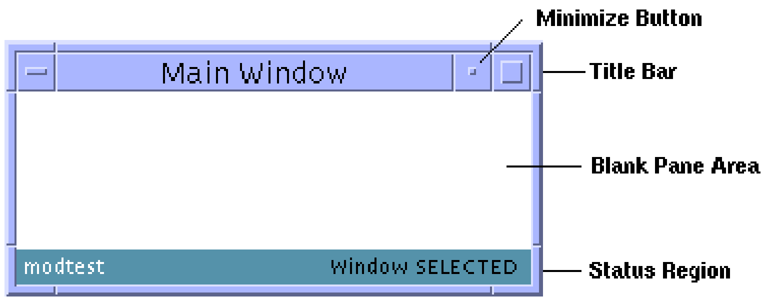
	Test Shown Modules is for testing help, menus, and connections in current, shown modules. All objects will be shown, including those for which the initial state is not set to Visible.
	Test Project is for testing help, menus, and connections in the current project. Objects for which the initial state is not set to Visible will not be shown.
Windows palette	Includes the three App Builder window objects: main window, custom dialog, and file selection dialog. Window objects are dropped on the desktop. See “Windows Palette” below for details.
Panes palette	Includes the four App Builder pane objects: control pane, text pane, draw area pane, and term pane. Pane objects are dropped on main windows, custom dialogs, or other panes. See “Panes Palette” on page 117 for details.
Controls palette	Includes 14 App Builder control objects: button, menu button, combo box, option menu, menu bar, radio box, check box, gauge, scale, separator, text field, label, list, and spin box. Control objects are dropped on control panes. See “Controls Palette” on page 119 for details.
Object information area	Provides information about the object beneath the cursor. See “Object Information Area” on page 121 for details.

Windows Palette

The Windows palette contains three objects: main window, custom dialog, and file selection dialog.

Main Window


Main Window Icon



Main Window Object

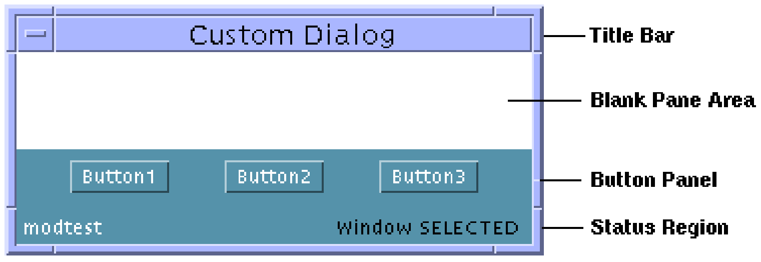
A *main window* is the basic App Builder object. It is created by dropping a main window icon on the desktop. The starting point for a user interface is built in a main window. A main window has a minimize button and therefore can be iconified.

The status region includes the name of the module the window is part of and indicates when the window object is selected. It does not appear in the compiled application.

Examples of main windows used in building App Builder itself are the App Builder primary window, the Project Organizer, the Module Browser, and the Code Generator.

Custom Dialog

Custom Dialog Icon



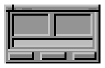
Custom Dialog Object

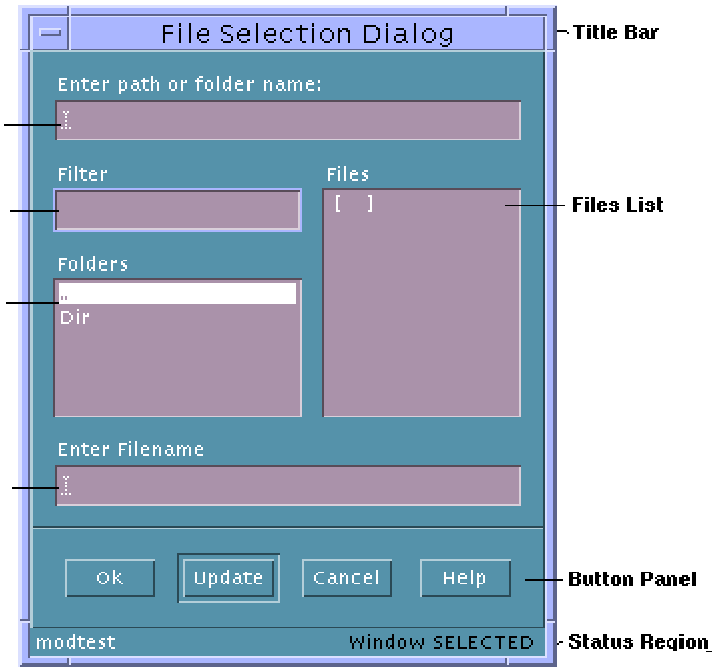
A *custom dialog* is a window for displaying information or providing a pop-up for a specific task within an interface. It is created by dropping a custom dialog icon on the desktop. A custom dialog might be "connected" to a button or a menu in a main window, causing the pop-up dialog to be displayed when the button is clicked or a menu item is chosen. A custom dialog cannot be closed to an icon.

The status region includes the name of the module the dialog is part of and indicates when the dialog object is selected. It does not appear in the compiled application.

Examples of custom dialogs used in building App Builder include the File Selection Dialog, the Project Name and Module Name dialog boxes, all of the editors, and the message dialog boxes.

File Selection Dialog


File Selection Dialog Icon



File Selection Dialog Object

A *file selection dialog* is a specialized pop-up dialog for specifying a file in an Open or Save operation. It is created by dropping a file selection dialog icon on the desktop.

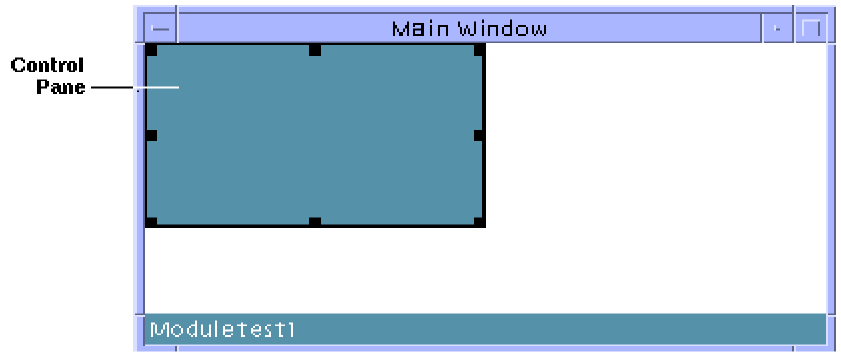
The status region includes the name of the module the dialog is part of and indicates when the dialog object is selected. It does not appear in the compiled application.

Panes Palette

The Panes palette contains four objects: control pane, text pane, draw area pane, and term pane. All panes can be dropped on a main window, a custom dialog, or another pane. If a pane is dropped on a pane, the dropped pane will become a child of the first pane or a layered pane will be created. See "To Create a Layered Pane" in Chapter 5, "Creating and Editing Panes, Menus, and Messages," for more information.

Control Pane

Control Pane Icon



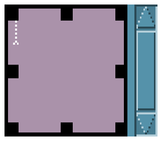
Control Pane Object

A *control pane* is the drop site for App Builder controls. It is created by dropping a control pane icon on a main window, a custom dialog, or another pane. In the figure above, a control pane has been dropped on the top-left corner of a main window, in anticipation of resizing it to fill the entire canvas.

Examples of control panes used in building App Builder include the pane on which the three panes palettes reside on the App Builder primary window and the pane beneath the controls on each of the property editors.

Text Pane

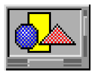
Text Pane Icon

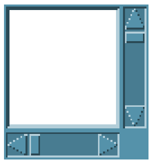

Text Pane Object

A *text pane* is a multi-line text-entry area in the completed application. It is created by dropping a text pane icon on a main window, custom dialog, or another pane.

Examples of the use of text panes in building App Builder include the Initial Value field in the Text Pane property editor and the Help Text field in the Help Editor.

Draw Area Pane


Draw Area Pane Icon


Draw Area Pane Object

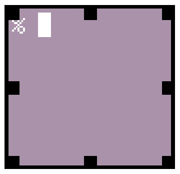
A *draw area pane* is used as a drawing or display area in the completed application. It is created by dropping a draw area pane icon on a main window, custom dialog, or another pane.

Note the horizontal and vertical scroll bars, which enable you to view objects outside the current view area.

Examples of the use of draw area panes in building App Builder include the panes displaying modules and module objects in the Module Browser, and the pane displaying modules in the Project Organizer.

Term Pane

Term Pane Icon


Term Pane Object

A *term pane* is a terminal emulation object which accepts user input and echoes standard output. It is created by dropping a term pane icon on a main window, custom dialog, or another pane.

Controls Palette

The Controls palette contains 14 objects, including buttons, lists, text fields, and a menu bar. To find out how to edit the properties of these objects, see Chapter 4, "Editing Properties of Interface Objects." To find out how to create menus and submenus and attach them to objects, see "Creating and Editing Menus" in Chapter 5, "Creating and Editing Panes, Menus, and Messages."

Button	A control which, when clicked, performs a specified action. A button can be a push button, a drawn button, or a menu button, settable in the Button property editor. A drawn button, like a push button, performs a specific function when clicked; the label on a drawn button, however, can change dynamically, depending on the status of the application.
Menu Button	A specialized button, ready for attachment of a menu. Note that there is no menu button property editor; edit the properties of a menu button in the Button Property Editor.

Combo Box	A combination text field and option menu object. As with an option menu, you can select an item from a pop-down menu, but you can also edit any of the items in the list—if you have checked "Editable" in the property editor, and if you write code to make it work.
Option Menu	One of the three "choice" objects (option menu, radio box, check box). When you click on an option menu, a menu is displayed, providing a choice of items to choose from. The chosen item remains in the option menu box and becomes the active choice. Examples of option menus in App Builder are Object Type in the property editors and Source and Target in the Connections Editor. An option menu is an exclusive-choice object.
Radio Box	One of the three "choice" objects (option menu, radio box, check box). A radio box is comprised of a label and two or more round buttons representing application functions, only one of which can be selected (hence the term "radio button," named for the type of buttons on an automobile radio). A radio box is an exclusive-choice object.
Check Box	One of the three "choice" objects (option menu, radio box, check box). A check box is comprised of a label and one or more check boxes, each with its own label. Each check box has a "binary" (on or off) state, and each is independent of the other. A check box is a nonexclusive-choice object.
Gauge	One of two "scale" objects (gauge, scale). A gauge is used to indicate a value.
Scale	One of two "scale" objects (gauge, scale). A scale, like a gauge, indicates a value, but a user can modify the value of a scale by moving the slider.
Separator	A horizontal or vertical line used to indicate separate functions in an application window.
Menu Bar	A horizontal bar of menu buttons arrayed across the top of a main window. The buttons are cascade buttons, for attaching menus. The default menu bar includes File, Edit, and Help topics. You can change,

	delete, or add to this group of topics. Note that the menu bar is not strictly a control object: it is a control pane with three buttons.
Text Field	A single-line text-entry area with a label (in contrast to a text *pane*, which is a is a multi-line text-entry area).
Label	A text string or graphic icon which can be attached to an object for identification purposes.
Scrolling List	An object for listing selectable options. A scrolling list is comprised of a variable-length list with scroll bars and an optional label. A list can allow single or multiple selections, and it can include a pop-up menu.
Spin Box	An object for selecting from a number of choices, only one of which is visible at any one time. A spin box is comprised of a text field, a label, and a set of arrows for sequencing through the choices.

Object Information Area

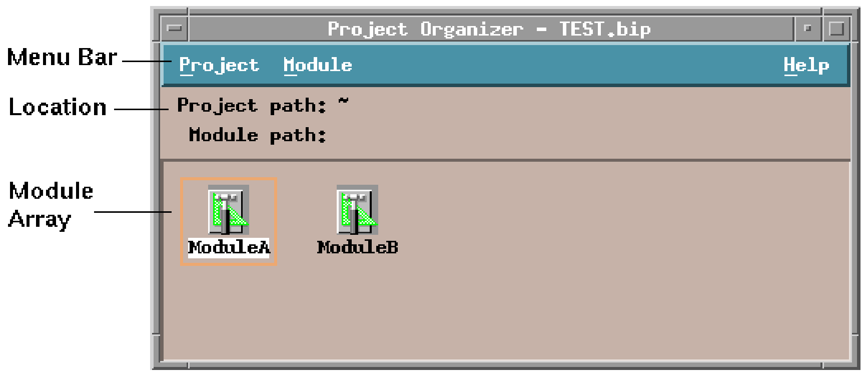
The object information area provides information about the object directly beneath the cursor—either on one of the primary window palettes or in the user interface. It includes the following information fields:

Object Type	The type of object beneath the cursor (main window, control pane, text field, for example). This field is active in the App Builder primary window, so you can use it to identify object types in the object palettes.
Object Name	The name of the interface object beneath the cursor. This name, in combination with the module name, uniquely identifies App Builder objects. Palette objects do not have names, so the field will be blank if the cursor is over the App Builder primary window. Note that all palette objects are given unique names when they are instantiated in the interface; you can change the name in the property editor for the object.
Position	The (x, y) pixel coordinates of the top-left corner of the object beneath the cursor, measured in the coordinate system of the object that contains it. If the object is a window object (main window, custom dialog, or file selection dialog), the position will be relative to the top-left corner of the monitor screen.

	If the object is a pane that was dropped on the top-left corner of a window, its position will be 0, 0, since 0, 0 are the coordinates of the top-left corner of the parent window. A pane that is dropped on another pane and made a layered pane also has coordinates of 0, 0. If the object is a control or a pane that has been made a child of a control pane, its coordinates are measured from the top-left corner of the parent object to the top-left corner of the child object.
Size	The size, in pixels, of the object beneath the cursor, in the form "width X, height Y."
Cursor Position	The (x, y) pixel coordinate location of the cursor, measured in the coordinate system of the object that contains it. Every object, including controls, has its own coordinate system. Some compound objects, comprised of more than one widget, have multiple coordinate systems; a custom dialog, for instance, includes a control pane, a tool bar, and buttons, each with its own coordinate system.
Editing Module	The name of the module currently being edited. Any window dragged from the Windows palette becomes part of that module. If more than one module is shown on the desktop, you can change the current module by selecting an object in another module.

Project Organizer

The Project Organizer is used to open, save, or close a project, and to save, show, hide, import, export, or remove modules.

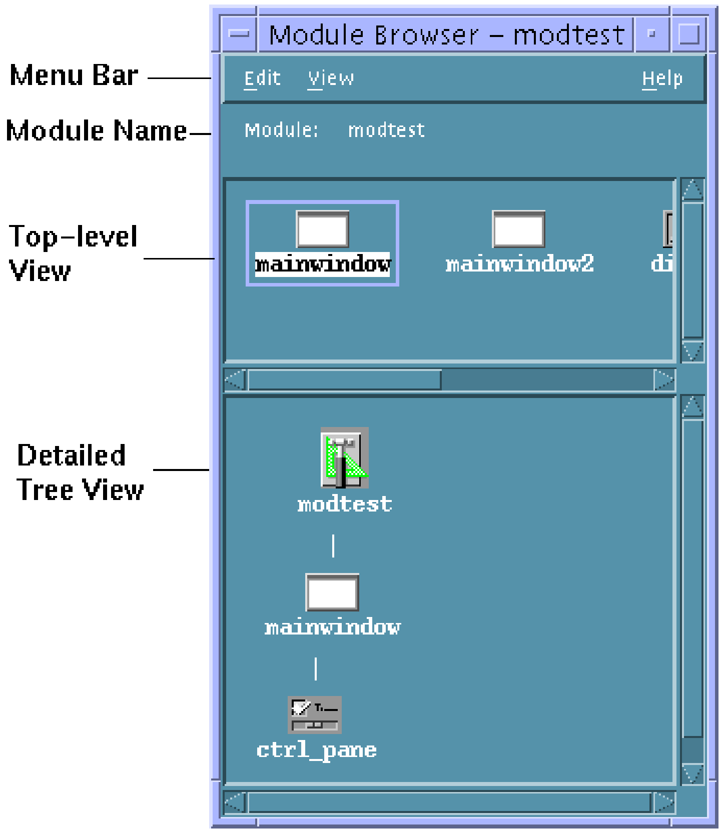


Project path: ~
Module path:

Location	A control pane with Project Path and Module Path fields; indicates the full-path location of the current project and the relative path to modules. The module will normally be in the same folder as the project, and its path will be noted as "." ("dot," signifying the current folder).
Module Array	A draw area pane that depicts each of the modules in the current project as a single icon with the module name beneath the App Builder icon.

Module Browser

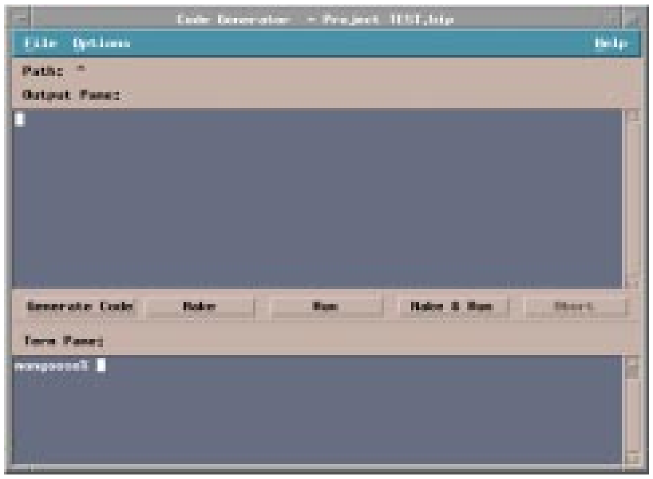
The Module Browser (also called the browser) provides a hierarchical, tree view of a module.



Module Name	Indicates the module being viewed. Can be changed through the View menu.
Top-level View	Shows all direct children of the module—windows, menus, and messages. A detailed view of each of the objects selected here is shown in the detailed tree view.
Detailed Tree View	Shows a detailed view of the top-level objects selected. All children of the top-level objects are shown.

Code Generator Window

The Code Generator window is used to generate code for the created interface and to make and run the completed application.

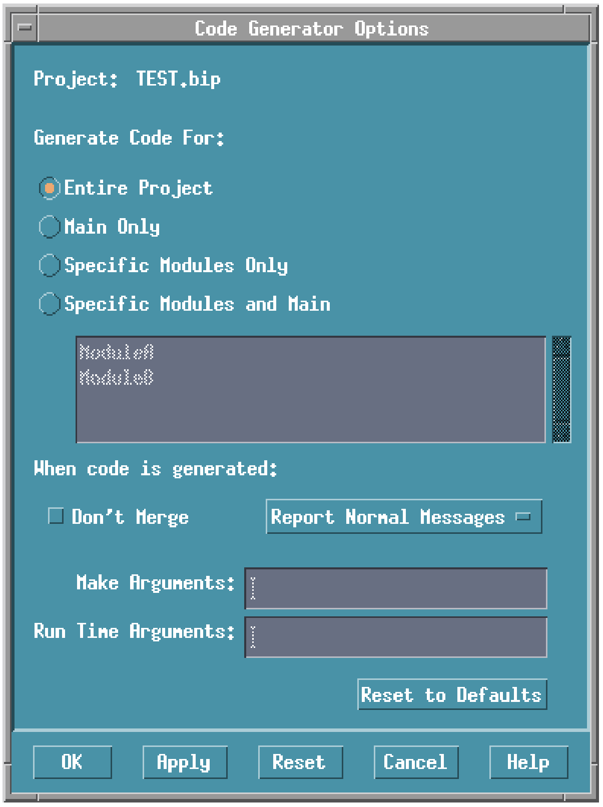


Path	Indicates the path to the current project, which is included in the title bar at the top of the window.
Output Pane	Refers to the text pane below this label. The results when you click on the buttons below the pane are displayed in this text pane. (The functions of the buttons also appear as menu items in the File menu.)
Generate Code	Generates code for the current project. The output for this action is displayed in the output pane.
Make	"Makes" the application for the current project. The output for this action is displayed in the Output Pane.
Run	Runs the compiled application after generating code and making the application. The output for this action is displayed in the Output Pane. The primary window for the compiled application will be displayed.
Make & Run	Combines the functions of the first three buttons (Generate Code, Make, Run). The output for this action is displayed in the Output Pane. The primary window for the compiled application will be displayed.

Abort	Aborts the currently running function. If the compiled application is being run, clicking Abort quits the application.
Term Pane	Performs any terminal emulation functions.

Code Generator Options Dialog Box

The Code Generator Options dialog box, accessible from the Options menu in the Code Generator window, is used to set options that determine what will happen when various Code Generator functions are performed.


Code Generator Options

Project: TEST.bip

Generate Code For:

- [✓ Entire Project]
- Main Only
- Specific Modules Only
- Specific Modules and Main

ModuleA
ModuleB

When code is generated:

☐ Don't Merge   Report Normal Messages

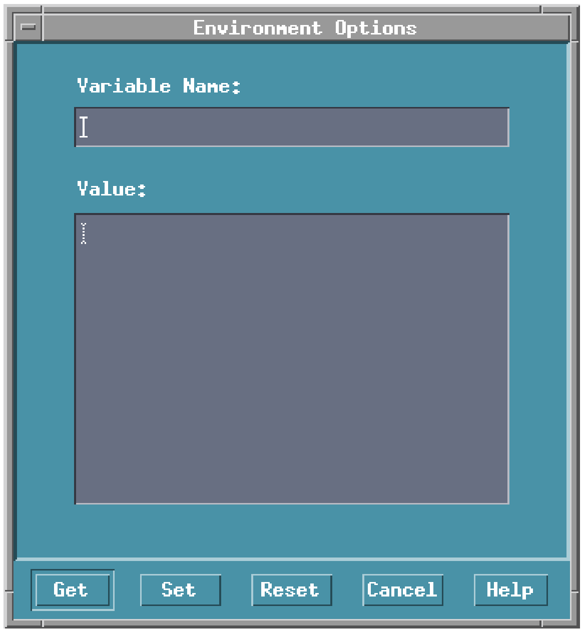
Make Arguments:

Run Time Arguments:

Project	The name of the current project.
Generate Code For	Specifies whether code will be generated for Entire Project, Main Only, Specific Modules Only, or Specific Modules and Main. If one of the latter two choices is specified, the modules in the scrolling list are active.
Don't Merge	Specifies whether user-written code will be merged into the generated code; if you check Don't Merge, any user-written code will be discarded when code is generated.
Report Normal Messages	Determines whether Normal Messages will be displayed in the output pane when code is generated, whether no messages will be generated (Be Silent), or whether all messages will be displayed (Be Verbose).
Make Arguments	Specifies what arguments will be appended to the Make command when it is run in the Code Generator.
Run Time Arguments	Specifies what arguments will be appended to the Run command when it is run in the Code Generator.
Reset to Defaults	Resets all Code Generator Options settings to their default values.

Code Generator Environment Options Dialog Box

The Code Generator Environment Options dialog box, accessible from the Options menu in the Code Generator window, is used for specifying a Variable Name and a Value for the variable, which value will be used for functions performed in the Code Generator window.



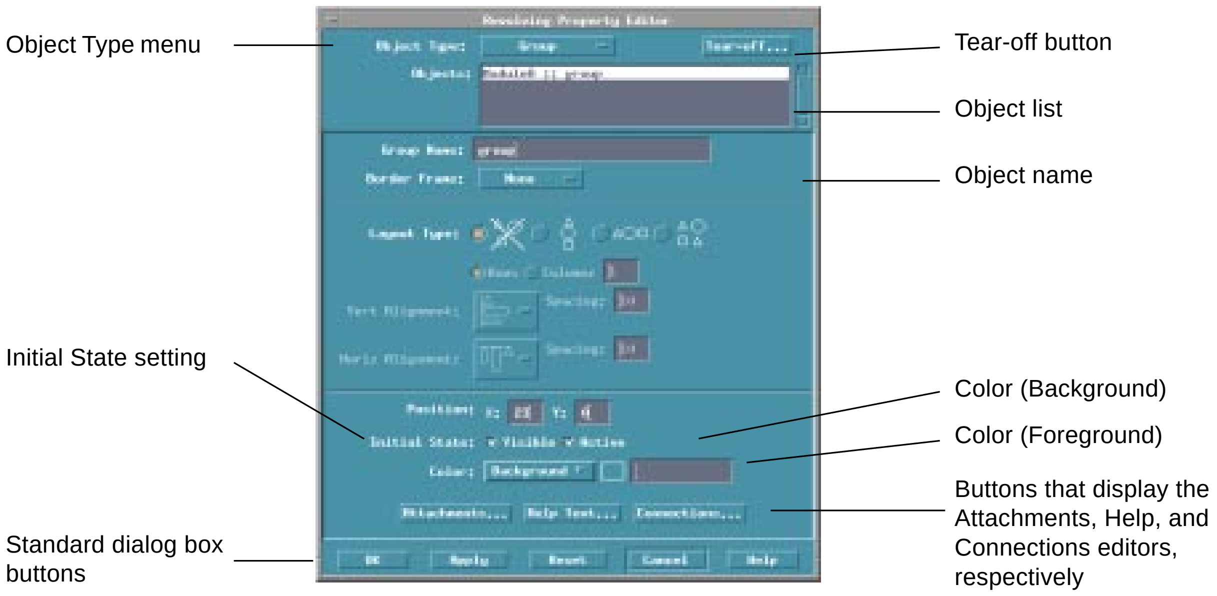
Variable Name	Specifies the name of an environment variable.
Value	Specifies a value for the variable specified in Variable Name. This value is only set for the Code Generator window and has no effect on the value of the variable outside of the Code Generator.
Get	Gets the current Code Generator value of Variable Name and displaying it in the Value text field.
Set	Sets Variable Name to the value in Value. This value is set for Code Generator window functions only.
Reset	Resets Value for Variable Name to its value as set outside of the Code Generator.

Cancel	Cancels any changes made to Value and closes the Environment Options dialog box.

Revolving Property Editor B

Used to edit the *properties* (the look or functionality) of interface objects, the Revolving Property Editor customizes your application interface. This appendix describes the properties common to all property editors, and the properties and the buttons common to a number of property editors. It also describes the individual property editors for each object.

The property editor for a separator, which includes the properties common to almost all property editors, is shown below, with common elements noted.



Property Editor: Universal Properties

The property editor for each of the App Builder objects is unique, but there are a number of properties common to almost all of them.

Object Type	An option menu for choosing the type of property editor to be displayed. Properties change depending on which object type is chosen.
Objects [list]	Lists the objects of the selected type in the current project. The list displays the full, unique name for each object, which is comprised of the name of the module in which the object exists, two colons, and the Object Name.
Object Name	Displays the default name or the name given by you to the object selected in the Objects list.

Initial State, Visible	Specifies whether the selected object is visible when the application starts up; all objects except a custom dialog are visible by default.
Initial State, Active	A check box for specifying whether the object selected is "active" when the application starts up. An inactive object is not functional: it is dimmed and no functions can be activated from the object.
Color: Background	Specifies the background color of the selected object. You can either type in a known color name or choose Color Chooser from the menu and select a color from the palette displayed.
Color: Foreground	Specifies the foreground color of the selected object. You can either type in a known color name or choose Color Chooser from the menu and select a color from the palette displayed.

Property Editor: Common Properties

The following properties are common to three or more property editors.

Border Frame	Determines the type of border, if any, around certain objects. Choices are None, Shadow Out, Shadow In, Etched Out, and Etched In.
Geometry	Indicates the X and Y location of the selected object, and the W(idth) and H(eight) of the object. X and Y values specify the position of the selected object in relation to its parent. The values are in pixels and are measured from the top-left corner of the parent object to the top-left corner of the child object. W and H values are in pixels.
Graphic Filename	Indicates the name of the pixmap (`.pm`) or bitmap (`.bm`) file that contains the graphic to be used as the label for the selected object or item. This property is available only if Label Type or Item Label Type is "Graphic."
Item Label Type	Specifies the type of label (String or Graphic) for the selected item in the Items list. If Graphic is chosen, "Label" becomes "Graphic Filename."
Item State, Active	Specifies whether the selected item will be active when the compiled application is opened.

Items	Lists the labels that represent the items in the list. When an item is selected in the Items list, its label is displayed in the Label or Graphic Filename field.
Label	Specifies the label for the selected object or item. "Label" becomes "Graphic Filename" if Graphic Label Type is chosen. Label is inactive in the Button property editor if Arrow Label Type is chosen.
Label Type	Specifies the type of label (String, Graphic, or Arrow) for the selected object. If Graphic is chosen, "Label" becomes "Graphic Filename." If you choose Arrow, the label in the Button property editor becomes an arrow and the Arrow Direction property becomes active.
Menu Title	Specifies the (optional) title of the pop-up menu, if any.
Popup, Pulldown Menu	A menu button and a text field for creating, attaching, de-attaching, or editing a pop-up or pull-down menu for the selected object. When the Menus button is clicked, a menu with four choices (None, Create New Menu, Menus, Edit Current) is displayed. Menus and Edit Current are inactive if no menus exist in the current project. If a menu is already attached to the selected object, the menu name will be displayed in the text field.
Position [Label]	Specifies the position (Left or Above) of the label in relation to the selected object. This Position option menu is next to the Label Type option menu.
Position [XY]	Indicates the X and Y location of the selected object in relation to its parent. The values are in pixels and are measured from the top-left corner of the parent object to the top-left corner of the child object.
Scrollbars	Specifies when scroll bars should be attached to the selected pane. The choices are Never and Always for a term pane or a text pane, and Never, When Needed, and Always for a draw area pane.
Size	Specifies the absolute W(idth) and H(eight) of the window or pane. These values change if you resize the window or pane manually in the interface. For a term pane or a text pane, there is an option menu for choosing Characters or Pixels as the unit value.

Size Policy	Specifies whether the selected object should retain a fixed size or if it should become bigger or smaller depending on the contents of the object. The choices are Size of Label and Fixed for buttons and labels, Fit Contents and Fixed for main windows and custom dialogs.

Property Editor: Common Buttons

The following functional push buttons or menu buttons are common to many property editors. The buttons at the bottom of the property editors (OK, Apply, Reset, Cancel, and Help) are common to other editors and dialog boxes.

Tear-off	Displays a property editor of the selected type; use this when you want to edit a specific object type while viewing other types of objects in the Revolving Property Editor.
Add Item	Adds an item after the selected item in the Items list. Added items are given default names starting with "Item1" and incrementing, as needed. By default, items are added after the selected item.
Edit	Performs edit functions (Add After, Add Before, Change, Cut, Copy, Paste, Delete) in a list. Add After and Add Before add an item to the list either after or before the selected item. Change applies the change that you have made. Cut, Copy, Paste and Delete act on the selected item, in the normal way: Cut and Copy place the selected item in a buffer, ready for Paste. Delete removes the item, but does not place it in a buffer.
Attachments	Displays the Attachments Editor; there is no Attachments button on the Main Window, Menubar, Custom Dialog, or Paned Window property editors.
Help Text	Displays the Help Editor.
Connections	Displays the Connections Editor.
OK	Applies the changes made to the selected object and dismisses the editor; changes to properties are marked with change bars at the left side of the editor.

Apply	Applies the changes made to the selected object, but does not dismiss the editor.
Reset	Resets all changes made since the last Apply.
Cancel	Resets all changes made since the last Apply and dismisses the editor.
Help	Displays help for the editor. See "Creating Help and Help Connections" on page 65 for information about App Builder help.

Individual Property Editors

An individual property editor is displayed by:

- Double-clicking an object in the interface or the browser.
- *Or,* selecting an object of the desired type and choosing Properties from the Editors menu on the App Builder primary window.
- *Or,* choosing Props from the interface or browser pop-up menu.
- *Or,* choosing the desired object type from the Object Type options menu at the top of the Revolving Property Editor.

The individual property editors are described in the following sections.

Button Property Editor

Only properties unique to a button object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Label Type, Label, Pulldown Menu, Size Policy, and Geometry.

Button Type	Specifies what kind of button (Push, Drawn, Menu) the selected button should be. Push button is the default. Selecting Menu transforms the push button into a menu button, as if you had dragged and dropped a menu button from the controls palette. The Pulldown Menu property becomes active if you select Menu. See "Controls Palette" in Appendix A, "App Builder Windows and Dialog Boxes," for descriptions of button types.

Label Alignment	Specifies the alignment (Left, Right, Centered) of the button label within the button border frame. Label Alignment is relevant only if Fixed is selected as Size Policy. This menu is inactive if Arrow Label Type is chosen.
Arrow Direction	Specifies which direction (Up, Down, Left, Right) the arrow should point if Arrow Label Type is chosen.

Choice Property Editor

Only properties unique to a choice object (Radio Box, Check Box, Option Menu) are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Label Type, [Label] Position, Label, Items, [Item] Label, Item State (Active), and Position [XY].

Choice Type	Specifies which type of choice object (Radio Box, Check Box, or Option Menu) the selected object should be. The object changes form depending on which you choose. Note that there is a control object for each of the choice types in the Controls palette. See “Controls Palette” in Appendix A, “App Builder Windows and Dialog Boxes,” for descriptions of choice types.
Rows/Columns	Specifies whether the radio box or check box should be laid out in rows or columns, and how many rows or columns there should be. Not relevant for an option menu.
Item State, Selected	Specifies whether the selected item will be selected when the compiled application is opened. Only one item can be selected. For a check box or a radio box object, the selected item will be marked as selected; for an option menu, the label for the selected object will be displayed in the option menu when the application is opened.

Combo Box Property Editor

Only properties unique to a combo box are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Label Type, [Label] Position, Label, Items, [Item] Label, and Position [XY].

Combo Box Type	Specifies whether the text field for the selected combo box will be Static or Editable in the compiled application. If Editable is selected, code must be written to implement the edit functionality.
[Label] Selected	Specifies which item will be selected when the compiled application is opened.
Width	Specifies whether the combo box shrinks or grows to accommodate the Longest Item in the list, or if the W(idth) of the box is Fixed. If Fixed is selected, the W(idth) value can be edited.

Control Pane Property Editor

There are no properties unique to a control pane. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Border Frame, Size Policy, Geometry, Popup Menu, and Menu Title.

Custom Dialog Property Editor

Only properties unique to a custom dialog object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Size Policy and Size.

Dialog Title	The title that appears at the top of the custom dialog.
Window Parent	An option menu for specifying a parent main window for the selected custom dialog. Choices are None and any main window in the project. If a main window is specified as a window parent, the custom dialog will

	be iconified and de-iconified with the main window. Note that this functionality does not work in test mode, but it does in the compiled application.
User Resize Mode	Specifies whether the window is Fixed or Adjustable (whether it can be resized in the compiled application).
Dialog Areas	Specifies whether a custom dialog includes a Button Panel (three buttons, by default) and a Footer area.
Default Button	Specifies one of the dialog buttons as the selected button, by default. The function represented by the selected button will be performed if you press Return while the mouse cursor is in the custom dialog in the compiled application.
Help Button	Specifies one of the dialog buttons as the help button. See “Creating Help and Help Connections” on page 65,” for a description of the Help Editor and instructions for creating help.

Draw Area Pane Property Editor

Only the one property unique to a draw area pane object is described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Scrollbars, Border Frame, Geometry, Popup Menu, and Menu Title.

Total Canvas Size	Specifies the W(idth) and H(eight) of the draw area canvas. Note that only a portion of the canvas will be visible if the draw area pane's size is smaller than the canvas size (400 by 400 pixels, by default). You can use the scroll bars to view other parts of the canvas.

File Selection Dialog Property Editor

Only properties unique to a file selection dialog object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color.

Window Parent	Specifies the main window parent of the file selection dialog. When displayed, the file selection dialog will appear over its main window. By default, the Primary Main Window is the parent of all file selection dialogs.

Dialog Title	Specifies the title that appears in the title bar at the top of the file selection dialog.
Initial Directory	Specifies the folder (directory) set as the starting value in the Path field of the file selection dialog.
Search Pattern Type	Specifies whether files, directories (folders), or both will be listed in the Files list of the file selection dialog.
Search Pattern	Specifies the value of the Filter field in the file selection dialog. The Filter value limits the files that will be listed in the Files field. The default value is * (asterisk), which means all files in the current folder will be listed. The Filter value for the Import Module file selection dialog in App Builder is `*.bil`, which means that only files that end in `.bil` will be listed.
OK Button Label	Specifies the label that will appear on the button in the left-most position at the bottom of the file selection dialog, normally labelled "OK." Clicking this button completes the file selection process and dismisses the file selection dialog. This button is labelled "Import" for the Import Module file selection dialog in App Builder.
Popdown Behavior	Specifies whether the file selection dialog will be automatically dismissed (the default) when the OK button is clicked.

Group Property Editor

Used to modify the layout and framing of groups, the Group Property Editor can be displayed by choosing Groups from the Editors menu of the App Builder primary window or by choosing Group from the Revolving Property Editor Object Type option menu. A group, unlike most of the objects edited in the Revolving Property Editor, is a created object and is not available from an object palette. See “Grouping Objects” on page 85,” for instructions.

Only properties unique to a group object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Border Frame and Position.

Note that choosing Groups from the Editors menu in the App Builder primary window is the same as clicking the Tear-off button in the Revolving Property Editor when the Object Type is Group.

Group Name	Displays the default name or the name given by you to the group selected in the Objects list.
Layout Type	Specifies As-Is, Vertical, Horizontal, or Row / Column layout of the objects in the selected group.
Rows Columns	Specifies whether the primary layout will be by rows (vertical layout) or columns (horizontal layout), and how many rows or columns to display. Active only if Layout Type is Row / Column.
Vert Alignment	Specifies left-edge, colon / label, center-line, or right-edge alignment of the objects in the selected group. Active only if Layout Type is Vertical or Row / Column.
Spacing	Specifies the number of pixels separating the objects in the selected group. Vert[ical] Alignment Spacing is active only if Layout Type is Vertical or Row / Column. Horiz[ontal] Alignment Spacing is active only if Layout Type is Horizontal or Row / Column.
Horiz Alignment	Specifies top-edge, center-line, or bottom-edge alignment of the objects in the selected group. Active only if Layout Type is Horizontal or Row / Column.

Label Property Editor

Only the property unique to a label object is described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Label Type, Label, Size Policy, and Geometry.

Note that no border appears around a label in the compiled application. See "To Create a Border around an Object" in Chapter 7, "Grouping and Attaching Objects," if you want a border around a label.

Label Alignment	Specifies the alignment (Left, Right, Centered) of the label within its margins. Label Alignment is relevant only if Fixed is selected as Size Policy.

List Property Editor

Only properties unique to a list object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Label Type, [Label] Position, Label, Items, [Item] Label, Position [XY], Popup Menu, and Menu Title.

Selection Mode	Specifies how objects can be selected in a scrolling list. Choices are Single Select, Browse Select, Multiple Select, and Browse Multiple Select. In Single Select mode, only one item can be selected, by clicking mouse button 1. In Browse Select mode, one item can be selected, but you can press mouse button 1 and drag through the list until the item you want is selected. In Multiple Select mode, you can make multiple, discontiguous selections with mouse button 1. In Browse Multiple Select mode, you can drag the cursor over items to make multiple, contiguous selections, and you can make a multiple, contiguous selection between a selected item and the cursor location with Shift-mouse button 1.
[Item] Selected	Specifies whether an item will be selected at application startup.
Width	Specifies whether the list object shrinks or grows to accommodate the Longest Item in the list, or if the W(idth) of the box is Fixed. If Fixed is selected, the W(idth) value can be edited.
Height	Specifies the number of text Lines in the list or its Pixels height.

Main Window Property Editor

Only properties unique to a main window object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State (Visible), and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Size Policy and Size.

Window Title	Specifies the title that appears at the top of the main window.
Icon File	Specifies the name of the graphics file that contains the graphical representation of the application icon—the object that is displayed when the application is "iconified" by clicking on the minimize button in the title bar.
Icon Mask File	Specifies the name of the graphics file that contains the bitmap that determines the shape of the visible representation of the icon beneath the icon mask. The icon mask acts like a stencil, allowing only the pixels in the icon that correspond to pixels in the mask to be visible.
Icon Label	Specifies the text label that appears beneath the application icon.
User Resize Mode	Specifies whether the window size is Fixed or Adjustable (whether it can be resized in the compiled application).
Window Areas	Specifies whether the main window will have a menu bar, a tool bar, or a footer. Note that a tool bar or a footer will show up as a control pane object in the Revolving Property Editor. You will probably want to add controls, such as the radio buttons in the App Builder primary window Build/Test tool bar, to a tool bar, and to make connections between the controls and programmatic functions. Code will have to be written to make a tool bar or footer functional.
Initial State, Iconic	Specifies whether the window is displayed as a window or an icon when the compiled application is opened.

Menu Property Editor

Used to create menus, the Menu Property Editor can be displayed by:

- Choosing Menus from the Editors menu of the App Builder primary window,

Note – Choosing Menus from the Editors menu in the App Builder primary window is the same as clicking the Tear-off button in the Revolving Property Editor when the Object Type is Menu.

- *Or*, choosing Menu from the Revolving Property Editor Object Type option menu,
- *Or*, choosing Create New Menu from the Popup Menu option menu in a property editor.

A menu, unlike most of the objects edited in the Revolving Property Editor, is a created object and is not available from an object palette.

Only properties unique to a menu object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Items, Label, and Item State (Active).

Add New Menu	Adds a new menu to the list of menus.
Edit	Performs edit functions (Cut, Copy, Paste, Delete) on the selected item in the list of menu objects. Cut and Copy place the selected item in a buffer, ready for Paste. Delete removes the item, but does not place it in a buffer.
Tearoff	Specifies whether tearoff is Enabled or Disabled. If tearoff is enabled the selected menu will be "postable." That is, the menu will be displayed until you explicitly dismiss it if you click on the Tearoff indicator (a dotted line).
Item Label Type	Specifies the type of label (String, Graphic, or Separator) for the item selected in the Items list. If Graphic is chosen, "Label" becomes "Graphic Filename." If Separator is chosen, Label or Graphic Filename becomes inactive and Line Style becomes

active. A Separator menu item is used to create a visual division in a menu, such as that seen in the Editors menu of the App Builder primary window.

Item Mnemonic	Specifies one of the letters in the selected item as a keyboard shortcut for choosing the item when the menu is posted. The letter specified will be underlined. Pressing the mnemonic letter when the menu is posted will cause that item to be chosen. Note that case is significant and that a particular letter can be used as a mnemonic only once within a menu.
[Item] Accelerator	Specifies a keyboard shortcut for choosing the selected item. An accelerator is comprised of a prefix (Ctrl, Alt, Meta, or Shift), an ampersand (&), and a letter (upper or lower case). To make Control-X an accelerator, for instance, type the following: `Ctrl&x` When you display the menu in test mode or in the compiled application, "Ctrl+x" will be included to the right of the menu item label. If you press the Control key and type x with the cursor in the window that contains the menu, the specified action will be performed. You can combine the Shift key with one of the other keys to form a compound prefix, if you wish. To make Shift Control-X an accelerator, type the following: `Shift Ctrl&x`
Line Style	Specifies the type of line style for the selected separator item; active only when Item Label Type is Separator. Choices are None, Etched In, Etched Out, Etched In Dash, Etched Out Dash, Single Line, Double Line, Single Dashed Line, and Double Dashed Line. A separator of the chosen line style will be displayed in the menu instead of a graphic or text label.
Item SubMenu	A menu button and a text field for attaching, de-attaching, creating, or editing a submenu for the selected item in the Items list. If a submenu is attached to the selected item, the name of the submenu will be displayed in the text field.

Menubar Property Editor

Only properties unique to a menu bar object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Items, Item Label Type, Label, Pulldown Menu, and Item State (Active).

Item Mnemonic	Specifies one of the letters in the selected item as a keyboard shortcut for displaying the menu. The letter specified will be underlined in the menu bar. In test mode and in the compiled application, the menu will be displayed if you hold down the Alt key and press the mnemonic letter while the mouse cursor is over the window that contains the menu bar.
Item State, Is Help Item	Specifies that the selected item is the Help menu. The Help menu appears at the right edge of the menu bar and has a built-in connection to the online help mechanism. The item labelled "Help" is the help button, by default.

Paned Window Property Editor

A paned window, unlike most of the objects edited in the Revolving Property Editor, is a created object and is not available from an object palette. See "To Create a Paned Window" on page 41 for instructions for creating a paned window.

Only properties unique to a paned window object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, and Initial State.

Panes	Lists the panes that comprise the paned window.
Pane Geometry	Displays the W(idth) and H(eight) of the pane selected in the Panes list.
Pane Height	Specifies the Min(imum) and Max(imum) height (in pixels) of the selected pane. These values determine the limits for the panes when you move the sash between panes.

Scale Property Editor

Only properties unique to a scale or gauge object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Label Type, [Label] Position, and Label.

Scale Type	Specifies Scale or Gauge. A scale includes a slider and is modifiable by a user (in the compiled application or in test mode). A gauge indicates a value, does not include a slider, and is not modifiable by a user.
Orientation	Specifies whether the scale object will be displayed in Horizontal or Vertical orientation.
Direction	Specifies Left to Right or Right to Left incrementing of value for a horizontal scale object, Bottom to Top or Top to Bottom incrementing of value for a vertical scale object.
Value Range	Specifies Min(imum), Max(imum, and Incr(ement) values for a scale object. All values must be integers. The increment value is used when you click with the mouse at either end of the scale object (in the compiled application or in test mode). See Decimal Points.
Decimal Points	Specifies the number of decimal places to shift the scale value when displaying it (if Show Value is checked). For example, a scale value of 250 with a Decimal Points value of 1 would display as 25.0; a scale value of 250 with a Decimal Points value of 2 would display as 2.50.
Show Value	Specifies whether the numerical value of the scale position will be displayed. See Decimal Points above.

Separator Property Editor

Only properties unique to a separator object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for a description of Geometry.

Orientation	Specifies whether the separator object will be displayed in Horizontal or Vertical orientation.

Line Style	Specifies the type of line style for the separator. Choices are None, Etched In, Etched Out, Etched In Dash, Etched Out Dash, Single Line, Double Line, Single Dashed Line, and Double Dashed Line.

Spin Box Property Editor

Only properties unique to a spin box object are described here. See “Property Editor: Universal Properties” on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See “Property Editor: Common Properties” on page 133 for descriptions of Label Type, [Label] Position, Label, Items, [Item] Label, and Geometry.

Spin Box Type	Specifies the type of spin box. If Numeric is chosen, the Items, Label, Add Item, Edit, and Selected properties are inactive. If String List is chosen, the Value Range, Initial Value, and Decimal Points properties are inactive.
Arrow Style	Specifies the style of arrow to be displayed on the spin box. Choices are Flat Beginning, Flat End, Beginning, End, and Split.
Value Range	Specifies Min(imum), Max(imum, and Incr(ement) values for a spin box object. All values must be integers. The increment value is used when you click with the mouse on one of the spin box arrows (in the compiled application or in test mode). Value Range is inactive if Spin Box Type is String List. See Decimal Points.
Initial Value	Specifies the starting value in the spin box in the compiled application. Initial Value is inactive if Spin Box Type is String List.
Decimal Points	Specifies the number of decimal places to shift the spin box value when displaying it. For example, a spin box value of 250 with a Decimal Points value of 1 would display as 25.0; a spin box value of 250 with a Decimal Points value of 2 would display as 2.50. Decimal Points is inactive if Spin Box Type is String List.
[Item], Selected	Specifies whether the selected item will be selected when the compiled application is opened. Only one item can be selected. Selected is inactive if Spin Box Type is Numeric.

Term Pane Property Editor

Only properties unique to a term pane object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Scrollbars, Border Frame, Position [XY], Size, Popup Menu, and Menu Title.

Process String	A text field for specifying the process (command) that will be run in the term pane in the compiled application. The default value is `/bin/csh`.

Text Field Property Editor

Only properties unique to a text field object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Label Type, [Label] Position, Label, Position [XY], and Width.

Operation	Specifies whether the text field in the compiled application and in test mode will be Editable or Read-Only.
Maximum Chars	Specifies the maximum number of characters that can be typed in the text field. This field is independent of the W(idth) field, which specifies the width of the displayed text.
Initial Value	Specifies the initial value to be displayed in the text field in the compiled application or in test mode.

Text Pane Property Editor

Only properties unique to a text pane object are described here. See "Property Editor: Universal Properties" on page 132 for descriptions of Object Type, Objects, Object Name, Initial State, and Color. See "Property Editor: Common Properties" on page 133 for descriptions of Border Frame, Position [XY], Size, Popup Menu, and Menu Title.

Operation	Specifies whether the text pane in the compiled application and in test mode will be Editable or Read-Only.

Word Wrap	Specifies whether words will be wrapped to the following line when the Size W(idth) value is reached.
Initial Value	Specifies the initial value to be displayed in the text pane in the compiled application or in test mode.

Index