Builder Xcessory 6.x Reference Manual

Integrated Computer Solutions Incorporated

Integrated Computer Solutions, Inc.

54 Middlesex Turnpike, Bedford, MA 01730

Tel: 617.621.0060

Fax: 617.621.9555

E-mail: info@ics.com

WWW: http://www.ics.com

Trademarks

Third printing

January 2009

Contents

Appendix C—License Manager

How to Use This Manual

Overview

The following table provides an overview of each chapter of the *Builder Xcessory Reference Manual*:

Chapter 1—Introduction	Describes how to start Builder Xcessory and provides a brief overview of Builder Xcessory main windows and system requirements.
Chapter 2—Main Window	Describes display modes, menus, and how to select objects on the Builder Xcessory Main Window and icons, and groups on the Builder Xcessory Palette.
Chapter 3—Resource Editor	Describes the menus of the Builder Xcessory Resource Editor, and how to update the Resource Editor.
Chapter 4—Managers	Describes the menu options and functions of the Builder Xcessory Style, Constant, Procedure, Identifier, Type, and UIL File Managers.
Chapter 5—Extended Editors	Describes the Builder Xcessory's extended editors, which facilitate the assignment of resource values to your interface objects.
Appendix A— Resources	Lists each user-settable resource in the Builder Xcessory.
Appendix B— Palette Objects	Lists class name, purpose, and features of each object on the Palette**.**
Appendix C— License Manager	Describes how to start and use the License Manager.

Notation Conventions

This document uses the following notation conventions:

{BX}

The syntax `{BX}` refers to the directory into which the Builder Xcessory is installed. The default directory is the following:

```
/opt/bxpro-6.0
```

Index

Most index entries are in lowercase:

fonts
  fixed width 28
  non-XLFD 228

Entries capitalized in Builder Xcessory are capitalized in the index:

Force To Selected 161
Force to Selected Tree 161

Languages

Because Builder Xcessory supports multiple programming languages, not all explanations or examples are applicable to all languages. The following icons indicate sections specific to particular languages:

C C **U** UIL **V** ViewKit **C++** C++ **J** Java

Note: Information that applies to all Motif environments does not use icons. In text, Motif refers to C, C++, ViewKit, and UIL.

Lists

The following two types of lists present procedures:

1. Numbered lists present steps to perform in sequence.

- Bulleted lists present alternate methods.

Objects

Objects are indicated as follows:

- Palette collection names are capitalized words with intercaps:

  Form or PushButton

- Instance names are lowercase words with intercaps:

  form or pushButton

- Class names are capitalized words with intercaps:

  Test or MyClass

Menu Notation

To indicate menus and menu options, the following format is sometimes used:

BX_window_name : menu_name : menu_item(or dialog_selection)

For example, Browser:File:Exit is the Exit selection from the File menu of the Browser window.

Text

- Literals such as file names, directory paths, code and text as typed on the command line are printed in `Courier` font:

  ```
  .bxrc6
  /usr/ics
  ```

- Text preceded by a % denotes text to enter at the command line:

  ```
  % bx
  ```

- Book titles, chapter names, and section names appear in *italic*:

  Builder Xcessory Reference Manual

  "Updating the Resource Editor" on page 136

- The main windows of the Builder Xcessory are capitalized as follows:

  Palette

  Browser

  Resource Editor

Definitions

This document uses the following terms:

Click	Move the cursor over an object, press a mouse button, and immediately release the mouse button. When the button is unspecified, assume mouse button one (typically the left mouse button).
Collection	A group of related user interface objects saved to the Builder Xcessory Palette for reuse. Collections can include any UI object supported by Builder Xcessory, including widgets, gadgets, C++ classes, or ViewKit components.
Component	A user interface object, generally used in the context of ViewKit classes. ViewKit components generally consist of collections of Motif widgets along with code to implement general or specific functionality, encapsulated into a C++ class subclassed from an element of the ViewKit application framework.
Cursor	A graphical image appearing on the screen which reacts to the movements of a mouse or other pointing device. In the Builder Xcessory, the cursor appears as an angle bracket when creating a widget, and an arrow when selecting a pull-down menu or menu item. During a drag and drop operation, it appears as an icon reflecting the type of object dragged and the target over which it is positioned.
Drag	Press a mouse button, then move the mouse without releasing the button. Typically followed with a drop operation. The phrase, “drag on to” indicates a drag and drop operation. Use MB2 to perform a drag and drop operation, unless otherwise specified.
Drop	Release the mouse button after positioning the mouse (and screen object) as desired. Typically follows a drag operation. The phrase, “drop on to” indicates a drag and drop operation. Use MB2 to perform a drag and drop operation, unless otherwise specified.
Enter	Type a new value and press the Enter key.
Gadget	A user interface object built upon the Xt Intrinsics (the X Toolkit). Similar to a widget, a gadget lacks certain data structures (such as a window) contained in a widget. The gadget maintains this data on the client side, rather than on the server, as does a widget. Although seldom used with today’s server performance levels, gadgets remain supported by BX.
{lang}	Specifies the currently selected language for code generation.

MB1, MB2 and MB3	Mouse buttons one, two, and three. Typically, MB1 is the left-most button, MB3, the right-most. On a two-button mouse, MB2 is most commonly emulated by pressing both buttons simultaneously. For actions described as “click,” assume MB1.
MB3 Quick Access menu	This menu is invoked by pressing MB3 while the mouse pointer is positioned over an object on the display. The contents of the menu depend on the type of object pointed to and the window in which you access the menu.
Object/ UI object	A reusable software entity with a user interface (UI), or visible, aspect. A generic term for the various objects that are manipulated with Builder Xcessory. UI objects include widgets, related collections of widgets, C++ classes, and ViewKit components. The term object and the term UI object are interchangeable.
Paste buffer	Cache into which a cut or copied object is placed. Also called a cut buffer.
Resize	To change the height and/or width of an object.
Resource	A user preference specification that controls elements of an application that can be customized by the user.
Select	To choose an object to be acted upon or an action to be performed; accomplished by clicking mouse button one on an object or menu item.
Session	A single, continuous working session, from the time you start Builder Xcessory from the command line, or from another tool, to the time you select Exit from the Browser’s File menu.
Widget	A user interface object built upon the Xt Intrinsics (the X Toolkit). Motif user interface objects are widgets.

Prerequisite Knowledge

This document assumes that you are familiar with the X Window System and OSF/Motif. If you are developing with ViewKit objects, this document assumes that you are familiar with these toolkits. Consult the following documentation lists for recommended references.

OSF/Motif documentation

For detailed descriptions of OSF/Motif and X, refer to the following documentation:

- *OSF/Motif™ Programmer's Reference*. Prentice-Hall, 1993. (ISBN 0-13-643115-1)
- *OSF/Motif™ Style Guide*. Prentice-Hall, 1993. (ISBN 0-13-643123-2)
- *OSF/Motif™ Programmer's Guide*. Prentice-Hall, 1993. (ISBN 0-13-643107-0)

X Window System documentation

- Asente, Paul, Donna Converse, and Ralph Swick. *X Window System Toolkit*. Digital Press, 1997. (ISBN 1-55558-178-1)
- Scheifler, Robert W. and James Gettys. *X Window System-Core Library and Standards*. Digital Press, 1996. (ISBN 1-55558-154-4)
- Scheifler, Robert W. and James Gettys. *X Window System-Extension Libraries*. Digital Press, 1997. (ISBN 1-55558-146-3)
- Scheifler, Robert W. and James Gettys. *X Window System-Core and Extension Protocols*. Digital Press, 1997. (ISBN 1-55558-148-X)

CDE documentation

For information about the Common Desktop Environment (CDE) widgets, refer to the following documents:

- *CDE 1.0 Programmer's Guide*. Addison-Wesley, 1995. (ISBN 0-201-48954-6)
- *CDE 1.0 User's Guide*. Addison-Wesley, 1995. (ISBN 0-201-48951-1)

ViewKit documentation

For a description of ViewKit (VKit) classes, refer to the following documentation:

- *ViewKit ObjectPak™2.1 Programmer's Guide*. Integrated Computer Solutions, 2002. (Included with the purchase of BX PRO.*)*
- *IRIS ViewKit™ Programmer's Guide*. Silicon Graphics, Inc. 1994. (Document Number 007-2124-002)

EPak documentation

For a description of EnhancementPak (EPak) widgets, refer to the following Integrated Computer Solution's documentation (included with BX PRO):

- *EnhancementPak™ 3.0 Programmer's Reference*. Integrated Computer Solutions, 1997.
- *GraphPak™ Programmer's Reference.* Integrated Computer Solutions, 1995.

Note: If you are using BX PRO, you can use and compile the EnhancementPak widgets and ViewKit objects in your interface. If you are using Builder Xcessory, you can use the EnhancementPak widgets and ViewKit objects in your interface, but you must purchase their respective libraries to compile any interface built with the EnhancementPak widgets or ViewKit objects. Contact your Sales Representative for more information.

Introduction

1

Overview

This chapter includes the following sections:

- **Starting Builder Xcessory** on page 2
- **Builder Xcessory Main Windows** on page 5

Starting Builder Xcessory

After installing Builder Xcessory on your system, enter the following command at the command line:

```
% {BX}/bin/bx
```

where {BX} is the directory where you installed Builder Xcessory.

If your system administrator copied the `bx60` script to a location in your search path, enter the following command at the command line:

```
% bx
```

Note: Refer to the *BX PRO Installation Notes* for complete installation instructions.

Startup Panel

While loading, Builder Xcessory displays the following Startup Panel:

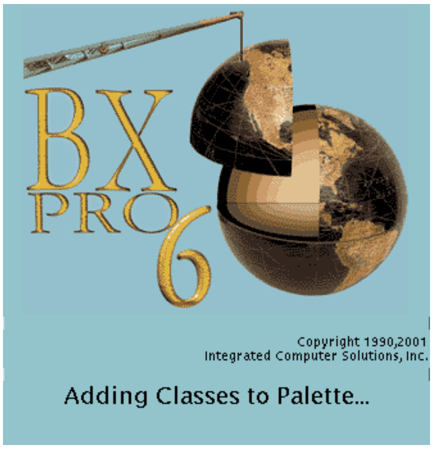

BX PRO 6
Copyright 1990,2001
Integrated Computer Solutions, Inc.
Adding Classes to Palette...

Figure 1. ICS Builder Xcessory Startup Panel

Startup Panel

The Startup Panel appears each time you start a Builder Xcessory session. As Builder Xcessory initializes, status messages are displayed at the bottom of the window, and a random tip is displayed in the “Did You Know” field. The “Next Tip” button is active only when the Builder Xcessory finishes loading (“System Ready” is displayed at the bottom of the window).

Closing the Startup Panel

Once Builder Xcessory is loaded, the Builder Xcessory main windows are displayed, with the Startup Panel in the foreground. Click on the Next Tip button to read more tips, or click on Dismiss to close the Startup Panel.

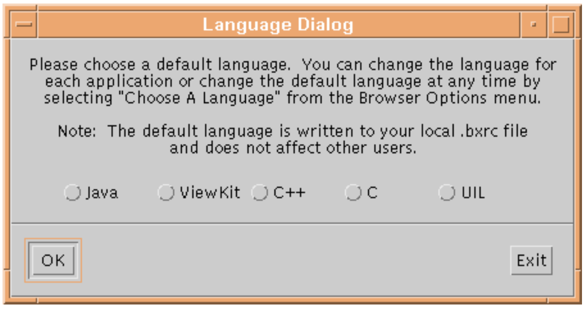
Dismissing the Startup Panel automatically

To dismiss the Startup Panel automatically when the system is ready, enable the Auto Dismiss Startup Panel toggle on the Behavior tab of the Browser Users Preference dialog.

Note: In your first Builder Xcessory session, the “Next Tip” button is not active until Builder Xcessory finishes loading and you select a default language. Refer to the next section, *“Language Dialog”* for more detailed information.

Language Dialog

In your first Builder Xcessory session, the following Language Dialog also appears in the foreground:



Please choose a default language. You can change the language for each application or change the default language at any time by selecting "Choose A Language" from the Browser Options menu.

Note: The default language is written to your local .bxrc file and does not affect other users.

Figure 2. Language Dialog

Selecting a default language

You must select a default language before proceeding. The language you select determines the available UI objects and the contents of the Builder Xcessory menus and dialogs. The selected language is saved in your `.bxrc6` file, and becomes the default language for all subsequent BX sessions.

You can change the language for each application or change the default language at any time by selecting Choose A Language from the Browser options menu. For more detailed information, refer to *“Choose A Language”* on page 59.

Exit button

The Exit button exits Builder Xcessory without saving a default language with your `.bxrc6` file. The Language Dialog will appear the next time you start Builder Xcessory.

Note: The Language Dialog appears during your first Builder Xcessory session only. Subsequent Builder Xcessory sessions use the default language, as specified in your `.bxrc6` file.

Note: When you select UIL as your default language, the Builder Xcessory utilizes the UIL save file to implement the interface and generates C code to implement the application.

Builder Xcessory Main Windows

Once you select a default language, you can access the two main windows of Builder Xcessory:

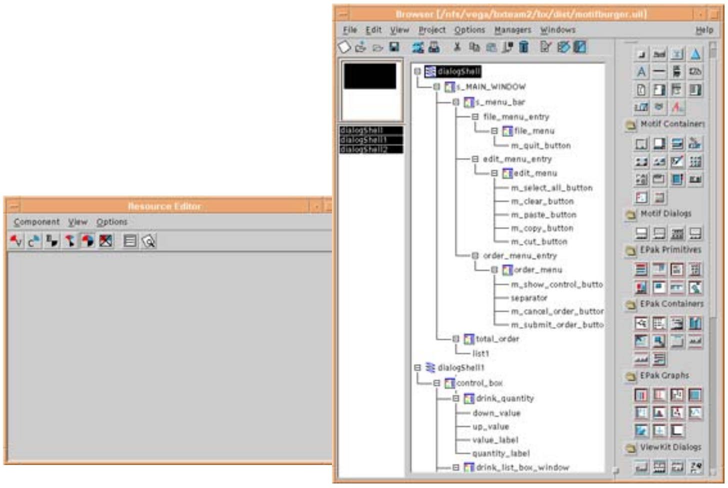


Figure 3. Builder Xcessory Main Windows

Tear-off menus

All Builder Xcessory menus are tear-off menus that you can manipulate as separate windows. With the menu displayed, select the dotted line at the top of the menu. Tear-off menus are iconified and de-iconified with their parent window.

Main Window

Browser

The Browser, the principle window used to control Builder Xcessory actions, displays the hierarchical structure of your interface.

Display modes

The Browser has the following two display modes:

- Instances View (the default)
- Classes View

Refer to *Chapter 2—Main Window* for a complete description of the Browser.

Palette

Depending upon the default language, the Palette displays labeled, iconic representations of the following objects:

- Entire set of Motif Xm interface widgets
- ViewKit ObjectPak Vk objects
- Subset of the EnhancementPak Xi widget set
- Platform-specific objects

The Palette can also include user-created collections of objects, other widget sets, and user-created classes.

Refer to *Chapter 2—Main Window* for a complete description of the Palette and to *Appendix A—Resources* for detailed information about the Palette icon objects.

BX PRO users

If you are running Builder Xcessory PRO, you can use and compile the EnhancementPak widgets and ViewKit objects in your interface.

Note: By default, the BX PRO Palette includes the EnhancementPak widgets.

Builder Xcessory users

If you are running Builder Xcessory (not BX PRO), you can use the EnhancementPak widgets and ViewKit objects in your interface, but you must purchase the respective libraries to compile any interface built with the EnhancementPak widgets or ViewKit objects. Contact your Sales Representative for more information.

Note: Start with EPak Widgets must be set on the Behavior tab of the User Preferences dialog to display the EnhancementPak icons. Refer to *“Behavior”* on page 96 for more detailed information.

Resource Editor

The Resource Editor displays the resources (such as height, sensitivity, background) of the currently selected object. You can set the Resource Editor to display all resources of the selected object, or some subset of these resources, by selecting the items on the Resources menu. The Resource Editor defaults to display the Simple Resources subset of resources.

Refer to *Chapter 3—Resource Editor* for a complete description of the Resource Editor.

Main Window

2

Overview

This chapter describes the following components of the Builder Xcessory Main Window:

Browser

Object Instance Hierarchy

Depending on the current display mode, the Browser displays the object instance hierarchy of your interface or the class instance hierarchy of your interface. From the Browser, you can accomplish the following operations:

Browser operations

- Select a class, class instance, or widget by clicking on its name in the Browser.
- Edit the structure of your interface by using drag and drop functions, the Browser Toolbar, and the Browser menus.
- Define new classes and display their contents.
- Generate C, C++, C/UIL, and ViewKit, and customize the names and contents of these files.
- Customize Builder Xcessory's integration with CodeCenter, ObjectCenter, Purify, XRunner, and RCS, SCCS, or your own source code control tools.
- Customize the generated code, including the contents of your makefiles.
- Access Builder Xcessory's online Help files.

Browser Display Modes

The Browser hierarchy has two display modes:

- Instances view (default)
- Classes view

The following sections describe the Browser display modes in more detail.

Browser in Instances View

In Instances View, the Browser displays the object instance hierarchy of your interface.

Entering Instances View

To enter Instances View, select Show Project Instances from the View menu.

The following figure displays an example of the Browser in Instances view:

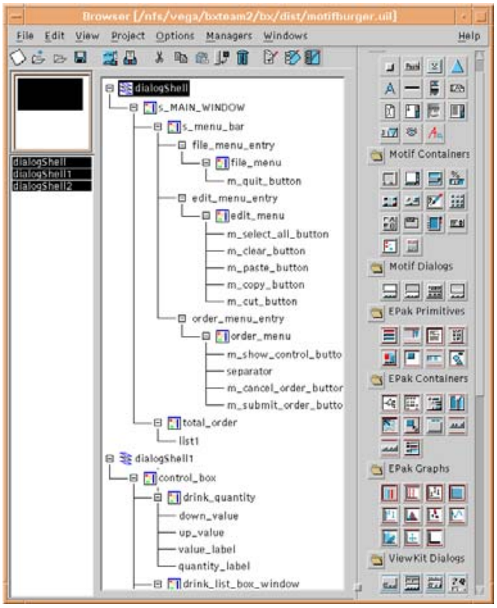


Figure 4. Object Instance Hierarchy

Displaying and hiding topLevelShells

A list of all topLevelShells in your interface is displayed below the Paner in a scrolled list. Click on a member of this list to toggle its display on the Browser. Highlighted members are displayed, and un-highlighted members are hidden.

Note: Hiding a topLevelShell decreases startup time and saves screen space (which facilitates browsing large interfaces). Hiding a topLevelShell does not affect code generation.

In Instances View, the Resource Editor displays different information depending on whether the currently selected object is a class instance or widget instance. Refer to *Chapter 3—Resource Editor* for more detailed information about the Resource Editor.

Browser in Classes View

In Classes View, the Browser displays the class instance hierarchy of your interface.

Entering Classes View

To enter Classes View, select Show Project Classes from the View menu. The Browser displays the object instance hierarchy of all classes in your interface:

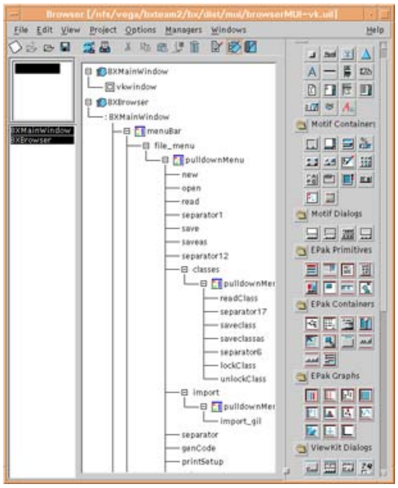


Figure 5. Class Instance Hierarchy

Changes that you apply to a class are applied to all instances of that class in your interface. Although you might not observe these changes until you return to Instances View, these changes happen immediately. For example, if you change the membership of a class and generate code while still in Classes View, the generated files reflect this change.

Note: If a class contains an object unavailable in the selected language, the class is not displayed. For example, a class which contains a Java object is not displayed if you selected C as your language.

Depending on whether the currently selected object is a class, widget instance, or class instance, the Resource Editor displays different information in Classes View. For more detailed information, refer to *Chapter 3—Resource Editor*.

Selecting Objects

Selecting objects

To select an object, use one of the following methods:

- Click on the instance name in the Browser.
- Click on the object itself on your display.
- Use the Browser Search/Select Bar.

In the following figure, the DrawingArea widget is selected:

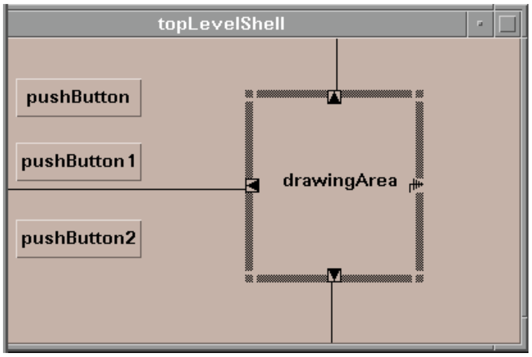


Figure 6. Selected Widget

The selected object is highlighted on the Browser display of the object instance hierarchy. The selected object is surrounded by a darkened selection outline on your interface. The selection outline is broken into corners and edges to aid in resize operations.

Selecting multiple objects

To select multiple objects, use one of the following methods:

- Hold down Ctrl and click on each object or instance name.
- Hold down Shift and MB1 and drag the cursor to draw a square around the objects (or their instance names). Release Shift and MB1. All objects completely within the square are selected.
- Hold down Shift and click on an object. The object and all objects descending from it are selected.
- Use the Browser Search/Select field.

Panning the Browser

XiPorthole and XiPanner

The EnhancementPak objects, XiPorthole and XiPanner, allow you to navigate more easily around the Browser object instance hierarchy display. The Panner widget represents a rectangular region (the canvas) of which only a smaller, enclosed rectangular region (the slider) is visible at any given time. It is used with a Porthole widget to scroll the Browser object instance hierarchy in two dimensions. The slider may be moved around the canvas by moving the cursor over the slider, pressing MB1, dragging, and then releasing the button.

Resizing the Panner

The Panner is fixed in the top left corner of the Browser. To resize, click and hold MB1 on the Browser and drag its resize border.

Automatic panning

Automatic panning is available when you use drag and drop to move or copy a widget or class instance on the Browser. Move the drag-and-drop icon on top of any border or corner of the Browser. The Browser display will pan in increments toward that direction. This allows you to pan automatically when you perform a drag-and-drop operation in the Browser. To stop panning, move the icon off the Browser border.

File Menu

Allows you to create new files, open files, merge other files with your project, save your interface, save and load classes, import GIL files, generate code, print, or exit Builder Xcessory. Select File from the Browser menu bar, either with the mouse or with a mnemonic, to display the following menu:

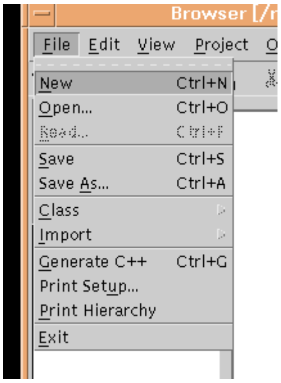
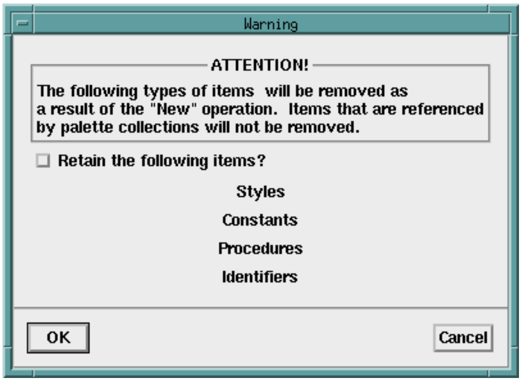


Figure 7. File Menu (with C++ as the Selected Language)

The following sections describe the items available from the File menu.

New (Ctrl+N)

Clears all current objects and switches to the default language. If any pixmaps, styles, procedures, identifiers, user-defined types or constants will be destroyed by the clear operation, Builder Xcessory displays the following warning message dialog:



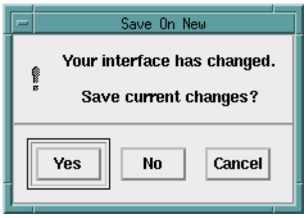
The following types of items will be removed as a result of the "New" operation. Items that are referenced by palette collections will not be removed.

☐ Retain the following items?

Styles
Constants
Procedures
Identifiers

Figure 8. Retain After New Warning Dialog

Retain If you set the Retain toggle, Builder Xcessory retains the listed items while clearing the interface. If your interface changed since your last save, the following dialog is displayed:



Your interface has changed.

Save current changes?

Figure 9. New Interface Warning Dialog

Clicking the Yes button prompts you to save your interface. Clicking the No button discards any changes to the interface since the last save. Click Cancel to return to Builder Xcessory and discontinue the new operation.

Open (Ctrl+O)

Note: Available only when an interface is not present on your display (for example, when you first startup Builder Xcessory, or when you select New).

Displays the File Selection dialog:

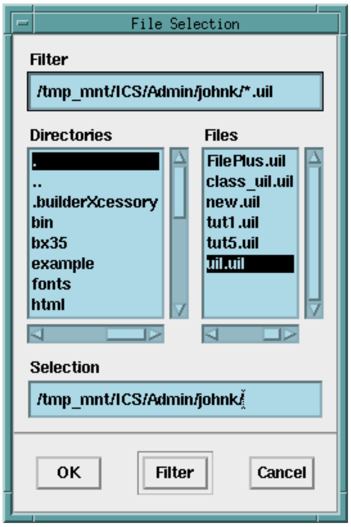


Figure 10. File Selection Dialog

Using the File Selection dialog

You can use the File Selection dialog to specify the following UIL files:

- UIL file generated by Builder Xcessory
- Existing UIL file written by hand
- Existing UIL files written with another user interface design tool

Note: When you open a UIL file not generated by Builder Xcessory 4.0 or later, the current language becomes the language associated with the UIL file.

Filter field

By default, the Filter field of the File Selection dialog contains the name of the directory from which you are running Builder Xcessory.

Directories box

To display the contents of a different directory, enter the directory name in the Filter field and click the Filter button. You can specify match strings using regular expressions. For example, `*.uil` will list only files ending in characters `.uil`. Click OK to open the specified file or Cancel to remove the File Selection dialog.

Files box The Files box lists the files in the directory which match the file filter, set to `*.uil` by default.

Selection field Enter the full pathname in this field to select a file.

Selecting a File To select a file, click on the file name in the File box or enter the full pathname in the Selection field.

If you open a file from a directory different from the one in which you started Builder Xcessory executable, Builder Xcessory changes to the current working directory.

Open imports a UIL file along with information specific to Builder Xcessory, such as generated code routine names and routine generation status. For example, if you change the Main file from `main-c.c` to `mymain.c`, this information is imported along with the file. If you disable code generation of a file, this information is also incorporated. However, this information is not imported during a Read operation.

Note: Builder Xcessory does not change to the new directory when you “Read” files.

Name conflict warning dialog When Builder Xcessory attempts to Open the selected file, it may display a warning dialog with the message “Name conflict between loaded style and new style”. A resource style that exists currently in the Builder Xcessory Style Manager has the same name as a different resource style defined for the file that you are attempting to open.

The three push buttons perform the following operations:

- Use Loaded

  Overrides the style in the file being opened with the information in the style that already exists in the Builder Xcessory Style Manager.

- Rename New

  Allows you to rename the style in the new file. Rename Style dialog is displayed. Enter the new style name in the text field. Click Cancel Read to cancel the opening of the new file without renaming any styles. Click OK to rename the style and remove the Rename Style dialog. If the new style name also conflicts with an existing style name, the Rename Style dialog is displayed again.

- Cancel Read

  Cancels the opening of the file. Has same function as Cancel Read push button on the Rename Style dialog. Any styles, constants, procedures, identifiers, and types in the new file are added to their respective managers. Refer to *Chapter 4—Managers* for more detailed information.

Read (Ctrl+F)

Displays the same File Selection dialog as shown in *“Open (Ctrl+O)”* on page 16. However, invoking the File Selection dialog from Read allows the following option and additional operations:

- You can add the interface specified in the UIL file to an existing interface on your display.
- Builder Xcessory classes, styles, constants, procedures, identifiers, and types in the new file are added to their respective managers.
- The contents of the read file are added to the current UIL file, and the read file is subsumed by the current UIL file.

Note: When Builder Xcessory attempts to Open the selected file, a warning dialog with the message “Name conflict between loaded style and new style” might be displayed. Refer to *“Name conflict warning dialog”* on page 17 for more detailed information.

Differences between Read and Open

Read differs from Open in the following ways:

- Read does not change the current working directory.
- All code generation options stored in the Read file are overridden by the current settings in Builder Xcessory.
- Read merges the UIL file into the currently opened project. Any association to the original UIL file is not maintained.

Save (Ctrl+S)

Writes your interface to the currently specified UIL file. Select one of the following options to display and edit the directory path and file name:

- Save As from the Browser File menu.
- Code Generation Preferences from the Browser Options menu.

Save As (Ctrl+A)

Displays a File Selection dialog:

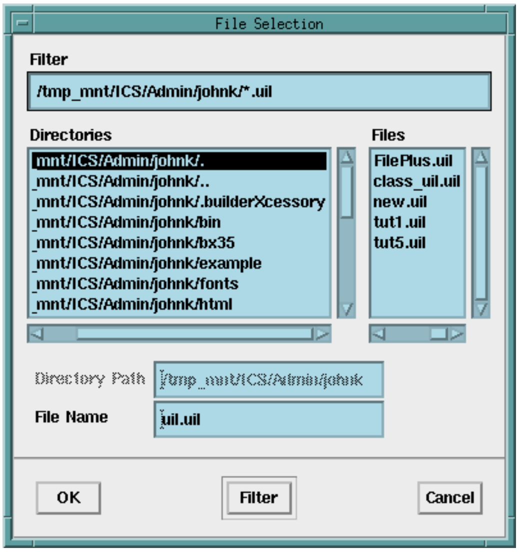


Figure 11. Save As File Selection Dialog

The Directory Path field is non-editable. You can rename the UIL file, or re-specify the directory path.

Editing the Filter text field

To edit the Filter text field, use one of the following methods:

- Double-click with the Directories list to navigate up and down directory paths.
- Click the Filter button to update the Directories list for the directory specified in the Directory Path field (which simply echoes what you specified in the Filter text field).

Note: Edits to the File Name text field change the name of the UIL file.

Class

Displays the following menu:

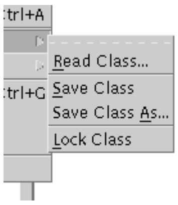


Figure 12. Class Option Menu from the File Menu

Read Class Reads in a class already created and saved as an include file by using the Save Class menu item.

When you select Read Class, a File Selection dialog is displayed. Specify the include file name or re-specify the directory path and click on the OK button in the File Selection dialog.

Builder Xcessory reads in the include file and adds it to the Palette.

Save Class Writes out a selected class in your application to a separate UIL include file. If you Save, the class definition is saved into an include file instead of the application’s UIL file, allowing you to easily add the class to other projects.

Note: The dialog displays the name of the class with `.uil` by default

Use Save Class when you have a class that you want to share with other developers or use in more than one application. Once you write a class to its own include file, other users can read that class into Builder Xcessory using Read Class.

Using Save Class

To use Save Class, use the following procedure:

1. Switch to Classes View in the Browser to display all current classes.
2. Select one class, then select Save Class.

   The first time you save a class, a File Selection dialog is displayed for you to specify the path and filename of the include file in which that class will be saved.
3. Enter a filename for the include file in the File Name text field at the bottom of the dialog, and click OK.

   The class is then written to the UIL include file you specified and can be read into Builder Xcessory by other users.

Loading a class automatically at startup

To save a class that will automatically be loaded into your Builder Xcessory at startup:

1. Select the class in Classes View
2. Select Save Class (see *"Save Class"* on page 20)
3. Drag icon of the class from its current location (probably Project Classes) on the Palette to a Private Classes group

Loading a class automatically at startup for a group

To save a class that will automatically be loaded into everyone's Builder Xcessory at startup, drag the icon of that class from its current location (probably Project Classes) on the Palette to a Public Classes group.

Once you save a class, you can rename the file using Save Class As, described in the following section.

Note: You must have write permission for the `{BX}/xcessory/classes` directory to drag the icon onto the Group. If you do not have write permission, ask someone with the proper permissions to move the file to the `{BX}/xcessory/classes` directory.

Save Class As

Displays a File Selection dialog:

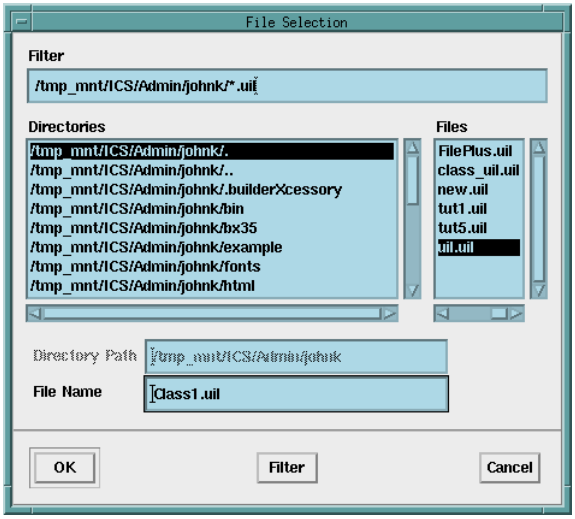

File Selection
Filter
/tmp_mnt/ICS/Admin/johnk/*.uil
Directories
/tmp_mnt/ICS/Admin/johnk/.
/tmp_mnt/ICS/Admin/johnk/..
/tmp_mnt/ICS/Admin/johnk/.builderXcessory
/tmp_mnt/ICS/Admin/johnk/bin
/tmp_mnt/ICS/Admin/johnk/bx35
/tmp_mnt/ICS/Admin/johnk/example
/tmp_mnt/ICS/Admin/johnk/fonts
/tmp_mnt/ICS/Admin/johnk/html
Files
FilePlus.uil
class_uil.uil
new.uil
tut1.uil
tut5.uil
uil.uil
Directory Path
/tmp_mnt/ICS/Admin/johnk
File Name
Class1.uil
OK
Filter
Cancel

Figure 13. File Selection Dialog

Note: The dialog displays the name of the class with .uil by default

You can rename the include file which contains a class that you saved separately from your application. You can also specify the directory path.

Lock Class

Makes the selected class read-only (you cannot edit the class). You must select a class before you click on this option. Builder Xcessory displays a warning dialog asking if you want to save the class before locking it:

1. Click OK to display a File Selection dialog (if you have not yet saved the class).
2. Enter the pathname of the file and click OK.

Note: Once you lock a class, a lock symbol is displayed in the Resource Editor when you click on any object in class. The lock symbol indicates that you cannot edit any attributes of the class or its constituent parts.

When you save your application, this file will not be overwritten.

Removing a lock To remove the lock:

1. Reselect the class.
2. Click on the Class option of the File menu.

The Lock Class option now reads Unlock Class. Select Unlock Class to remove the lock.

Note: The Browser must be in Classes View to enable the Lock Class feature. See *“Browser in Classes View”* on page 12 for more information.

Import

Displays a menu that contains only the Import Gil option.

Import GIL Displays the same File Selection dialog as shown in Open (Ctrl+O) and Read (Ctrl+F). See *“Open (Ctrl+O)”* on page 16 for more detailed information about the File Selection dialog. The File Selection dialog filter defaults to a `.G` file suffix.

The File Selection dialog allows you to add the interface specified in a GIL (Guide Interface Language) file to an existing interface on your display. You can set certain options before importing a GIL file. See the section *“GIL Import Preferences”* on page 109 for more detailed information.

Note: Builder Xcessory does not generate GIL files.This option is intended as an aid for OPEN LOOK to Motif interface conversions.

Generate {Lang} (Ctrl+G)

Writes the files for your application. `{Lang}` is the language you selected for this particular project file, or the language you selected as your default language at the beginning of your Builder Xcessory session. If you want to choose a different language, refer to *“Choose A Language”* on page 58 for more detailed instructions.

Once you save your application, you can generate your application code outside Builder Xcessory in scripts or makefiles with the following command on the command line:

```
cgx60 -gen {lang} -path {BX}/XCESSORY -file {file.uil}
```

The following sections describe how files are generated in each available language.

Generate C++

Writes the C++ files for your application. Additional files are generated if you included classes in your interface. See *“Code Generation Preferences”* on page 59 for more detailed information about filenames and options for the files to be generated.

Note: Builder Xcessory only generates files which have changed since the last code generation. For example, for the class MyClass, the files `MyClass.C` and `MyClass.h` are regenerated only if they are different from the previous code generation.

Files generated for C++

The following list describes the C++ files generated by Builder Xcessory when you select Generate C++, or execute the `cgx60 -gen CXX` command on the command line (default names are in parentheses):

- Imakefile (`Imakefile`)
  Imakefile that specifies C++ imake information.
- Makefile (`makefile-C`)
  C++ makefile.
- Class Tree Root (`UIComponent`)
  Contains the root class from which all C++ classes created in Builder Xcessory are derived.
- Main (`main-C.C`)
  Initializes the Motif toolkit and calls the Widget creation routine. Builder Xcessory creates an unrealized applicationShell and creates all topLevelShell children for all top-level windows.
- Constants (`defs-C.h`)
  Contains pixmaps, declarations of constants, and global instances of widgets or classes used in the interface.
- Utilities (`bxutils-C.C`)
  Contains convert functions.
- Callbacks (`callbacks-C`)
  Contains any callback information for non-class widgets.
- App Defaults (`app-defaults`)
  File that includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit
- A header file and source file for every class in the interface

Generate ViewKit

Writes the C++ files that use ViewKit classes for your application.

To generate your application code outside Builder Xcessory, use the following flag in the `cgx60` command on the command line:

```
cgx60 -gen VK <filename>
```

Files generated for ViewKit

The following list describes the ViewKit files generated by Builder Xcessory when you select Generate ViewKit, or execute the `cgx60 -gen VK` command on the command line (default names are in parentheses):

- Imakefile (`Imakefile`)
  Imakefile that specifies ViewKit imake information.
- Makefile (`makefile-vk`)
  ViewKit makefile for the current platform.
- Main (main-vk)
  Initializes the toolkit and creates the Window classes.
- Constants (`defs-vk`)
  Contains any pixmaps, declarations of constants, and global instances of widgets or classes used in the interface.
- Utilities (`bxutils-vk`)
  Contains convert functions.
- Callbacks (`callbacks-vk`)
  Contains any callback information for non-class widgets.
- App Defaults (`app-defaults`)
  File that includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or using the `-xrm` flag on the applications command line.
- A header file and source file for every class in the interface.

Generate C

Writes the C files for your application. To generate your application code outside Builder Xcessory, use the following flag in the `cgx` command:

```
cgx60 -genC -file > filename>
```

Files generated for C

The following list describes the C files generated by Builder Xcessory when you select Generate C, or execute the `cgx60 -gen C` command on the command line (default names are in parentheses):

- Imakefile (`Imakefile`)
  Imakefile that specifies C imake information.
- Makefile (`makefile-c`)
  C makefile for the current platform.
- Main (`main-c.c`)
  Initializes the toolkit and calls the Widget Creation routine.
- Creation (`creation-c.c`)
  Contains code to create the widget hierarchy. Functions in this file are called by the Main file.
- Callbacks (`callbacks-c.c`)
  Contains any callback stubs for any callbacks defined in Builder Xcessory. When you write out the Callbacks file to a directory with an existing file of the same name, Builder Xcessory scans the existing file for the names of functions, and appends callback stubs for undefined callbacks only. Callbacks are never removed from this file and are never overwritten. Stubs for timers, event handlers, and work procedures are also output in this file.
- Constants (`creation-c.h`)
  Contains any pixmaps, declarations of constants, and global instances of widgets or classes used in the interface.
- Utilities (`bxutil-c.c`)
  Contains convert functions, predefined callback code, and required xpm functions.
- App Defaults (`app-defaults`)
  Includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit.

Generate UIL

Generates code for your application using a combination of C and UIL. Builder Xcessory generates your interface in a set of UIL files specified with the UIL File Manager. Other files are generated to compile, invoke, and run your UIL from outside Builder Xcessory.

Files generated for UIL

The following list describes the UIL files generated by Builder Xcessory when you select Generate UIL, or execute the `cgx60 -gen C_UIL` command (default names are in parentheses):

- UIL File (`uil.uil`)

  Contains a description of the widgets. Compiled to a `.uid` file which is then read by the Mrm calls at run-time to create the interface. Builder Xcessory generates a backup file whenever it overwrites an existing UIL file. Automatic backup is provided for the default filename `uil.uil` or any customized filename. If, for example, you write the renamed UIL file `new_uil.uil`, and a copy of this file already exists, Builder Xcessory generates the backup file `new_uil.uil~`. These UIL files are identical to the files generated when you select Save from the Browser File menu.

- Callbacks (`callbacks-uil.c`)

  Contains callback stubs. When you write out the Callbacks file to a directory with an existing file of the same name, Builder Xcessory scans the existing file for the names of functions and appends callback stubs for undefined callbacks only. Callbacks are never removed from this file and are never overwritten. Stubs for timers, event handlers, and work procedures are also output in this file.

- Constants (`main-uil.h`)

  Contains declarations of constants, and global instances of widgets or classes used in the interface.

- Utilities (`bxutil-uil.c`)

  Contains convert functions and predefined callback code.

- App Defaults (`app-defaults`)

  File that includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit

- Imakefile (`Imakefile`)
- Imakefile that specifies both UIL and C imake information.Makefile (`makefile-uil`)

  Makefile for the current platform. Has targets for UIL as well as the associated C files.
- Main (`main-uil.c`)

  Initializes the toolkit, initializes Mrm, and makes Mrm calls in order to create widgets. This file is intended as a template to enable you to immediately compile the interface output from Builder Xcessory, it will not be adequate for all applications. In particular, Builder Xcessory creates an unrealized applicationShell and creates all topLevelShell children for all top-level windows, as recommended in Asente, Converse, and Swick's *X Window System Toolkit*.

You can change any filename, and disable or enable the generation of a particular file from the File Names tab of the Code Generation Preferences dialog (Browser Options menu). Any changes made in the Output File Names dialog will be reflected in subsequent generation of code.

Note: You can modify this file within the specified user code blocks. However, changes made outside these blocks are not preserved between code generation sessions.

Print Setup

Displays the Print Setup dialog:

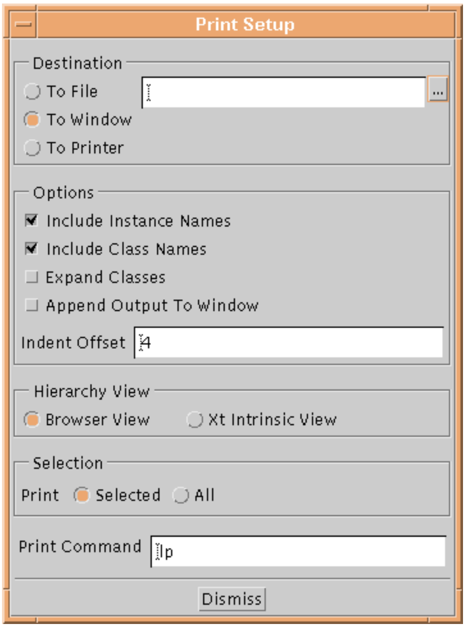


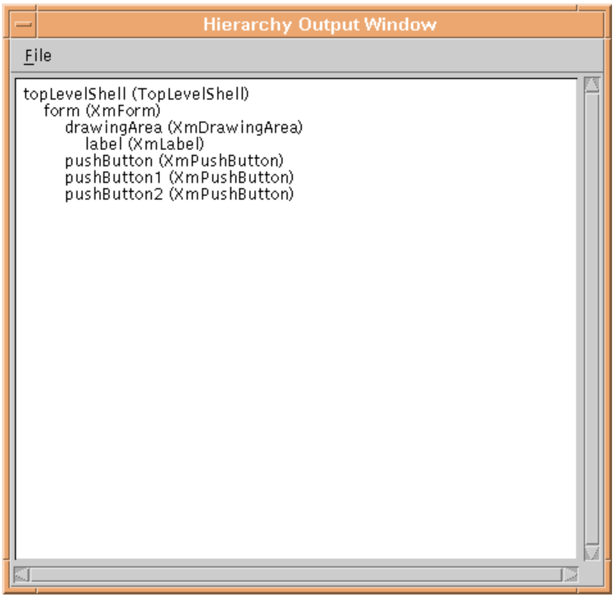
Figure 14. Print Setup Dialog

The toggles set on this dialog specify the options which are used when you select Print Hierarchy from the Browser File menu.

Destination

The radio buttons in the Destination field adjust whether the output of Print Hierarchy will go to a file, a viewer window, or a printer.

- To File
  Allows you to print to a file. Enter the name of the file or select the name in a file selection box accessed by clicking the (...) button.
- To Window
  Allows you to print to a viewer window. When you select Print Hierarchy, the print viewer window is displayed and the hierarchy is printed to that window:



```
topLevelShell (TopLevelShell)
   form (XmForm)
      drawingArea (XmDrawingArea)
         label (XmLabel)
      pushButton (XmPushButton)
      pushButton1 (XmPushButton)
      pushButton2 (XmPushButton)
```

Figure 15. Print Viewer

Clear this window, save as an ASCII file, or print it.

- To Printer

  Allows you to print a hierarchy using the Print Command specified at the bottom of the dialog.

Options

The toggles in the Options field adjust the format and content of the print output:

- Include Instance Names
  Includes the name of each object instance in the output.
- Include Class Names
  Includes the class name of each object instance in the output.
- Expand Classes
  Includes information for the full membership of classes in the output; if unset, only information about the class itself is included.
- Append Output to Window
  Specifies not to clear the viewer between outputs. If unset, the viewer will be overwritten with each output.
- Indent Offset
  Number of spaces each level of the hierarchy is indented (text field).

Hierarchy View

The toggles in Hierarchy View allow you to print either the Browser or the XtIntrinsic view of the widget hierarchy:

- Browser View
  Mirrors the contents of the Browser, showing the logical hierarchy of the objects in the interface.
- Xt Intrinsic View
  Shows the literal hierarchy from the Xt perspective.

Example

For example, the Browser hierarchy for a menu bar with a pulldown menu child is the following:

```
topLevelShell (TopLevelShell)
    menuBar (XmMenuBar)
        cascadeButton (XmCascadeButton)
            pulldownMenu (XmPulldownMenu)
                    pushButton (XmPushButton)
```

The Xt hierarchy for the same collection is the following:

```
topLevelShell (TopLevelShell)
    menuBar (XmRowColumn)
        cascadeButton (XmCascadeButton)
        pulldownMenuMenuShell (XmMenuShell)
            pulldownMenu (XmRowColumn)
```

```
pushButton (XmPushButton)
```

Selection

- Selected
  Allows you to print only the tree whose root is the currently selected object.
- All
  Allows you to print the entire interface.

Print Command Specifies the printer command appropriate for your system.

Print Hierarchy

Executes the print command as specified by the options in the Print Setup dialog.

Exit

Exits the Builder Xcessory. If you have made changes to your interface since last writing out the UIL file, Builder Xcessory displays a warning dialog that allows you to save your changes, quit your changes, or cancel the exit operation.

Note: Exiting Builder Xcessory does not initiate an exit of any development environment tools started from Builder Xcessory.

Edit Menu

The Browser Edit menu allows you to cut, copy, add an interface object to the Palette, paste, delete, raise or lower instances on the display, resize an instance to its natural size, make an instance larger, align multiply-selected widget instances, make (and unmake) classes, and designate a widget instance in a class as a receptor.

Select Edit from the Browser menu bar to display the following menu:

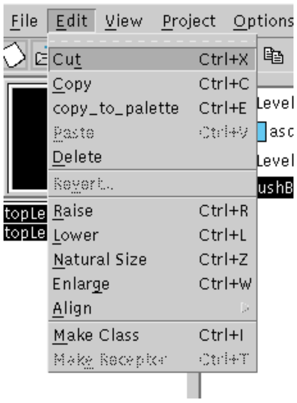


Figure 16. Edit Menu

You can also select several object instances, and execute Cut, Copy, Paste, or Delete on that group as if it were a single widget instance.

Cut (Ctrl+X)

Removes the selected instance and its descendants from the display and places them along with their resources in the instance paste buffer, overwriting any existing contents of the buffer. Use Cut with Paste to move the new instance or collection to a new location. You may Cut and Paste an object from one work space to another.

Note: You can paste only the instance most recently cut or copied.

Using drag and drop to cut and paste

Use drag and drop to Cut and Paste objects in the object instance hierarchy tree. Dropping instance name A on top of instance name B will Cut instance A and all of its descendants, along with their resources, from the interface and Paste them as descendants of widget B.

Dropping an instance name on an empty portion of the object instance hierarchy in the Browser display will Cut the instance and all of its descendants, along with their resources, from the interface and Paste the instance as the first child of a new topLevelShell. Dragging and dropping in this manner does not affect the instance paste buffer.

Note: The instance paste buffer and style paste buffer are entirely separate. For example, a Cut operation on a resource style will not affect the contents of the instance paste buffer.

Deleting groups

To cut several instances, you can save time by selecting those instances as a group and cutting the entire group, instead of cutting each individual instance.

When you cut multiple widget instances, each widget and its descendents are placed in the cut buffer. If the widgets are all from different widget trees (for example, different top level shells), they are placed in the buffer in the order selected. When you Paste a group of instances, those instances are pasted in the same order in which you selected them.

You can drag and drop either a single widget instance or a group of widget instances.

Copy (Ctrl+C)

Places a copy of the selected instance and its descendants, along with their resources, in the instance paste buffer, overwriting any existing contents of the buffer. Use Copy with Paste to copy the contents of the instance paste buffer in a new location. You may Copy an instance from one work space to another.

Using drag and drop to copy and paste

Use one of the following drag and drop methods to copy and paste objects in the instance hierarchy displayed as a tree:

- Drop instance name A on top of instance name B while depressing the Ctrl key to copy instance A and all of its descendants (along with their resources) from the interface, and Paste them as descendants of instance B.
- Drop an instance name on an empty portion of the object instance hierarchy while depressing the Ctrl key to Copy the instance as the first child of a new topLevelShell.

These drag and drop operations do not affect the instance paste buffer.

Copying groups

To save time, you can select several widget instances and then copy the entire group. Just as when you cut a group, each widget instance and its descendents are placed in the cut buffer. If the widgets are all from different widget trees, they are placed in the buffer in the order selected.

Note: When you Paste a group, instances are pasted in the same order in which you selected them.

Copy To Palette (Ctrl+E)

Copies the selected instance and all of its descendants, along with their resources, to the User Collections group on the Palette.

Instances and collections which are copied to the Palette will persist, with all their resources, from session to session until specifically removed.

Using drag and drop for Palette additions

You can also add instances and collections to the Palette using drag and drop, as follows:

1. Position the cursor over either the instance or its instance name on the Browser.
2. Press MB2.
3. While continuing to hold down the mouse button, drag the object to the Palette and drop it onto the desired group.

Paste (Ctrl+V)

Places the contents of the instance paste buffer. An outline of the contents of the instance paste buffer appears. Add this collection to the interface by placing the collection in the desired position. Use Paste in conjunction with Copy to copy an object, or with Cut to move an object.

Note: Because Paste does not clear the instance paste buffer, multiple Paste operations are possible.

When you paste a group of widget instances, each instance (and its descendents) is pasted in the order in which you selected it. The result is the same as cutting and pasting each instance separately.

Delete

Permanently removes the selected instance and all of its descendants. Delete does not copy the object to the instance paste buffer. By default, a confirmation box is displayed for the Delete operation. To turn the confirm delete prompt off, use the User Preference option from the Browser Options menu. See *“Behavior”* on page 91 for more detailed information about this option.

Note: Deleted instances are not recoverable unless they are specified in a UIL file which can be read back into Builder Xcessory.

When you delete a group of widget instances, each instance (and its descendents) is deleted in the order in which you selected it. The result is the same as deleting each instance separately.

Revert

Reverts the UIL file to a previously saved version, or the last autosaved version.

Hint: You can change the number of button and key actions between automatic back-ups by editing the Autosave Interval field on the General tab of the User Preferences dialog. Refer to section *“General”* on page 90 for more detailed information.

Raise (Ctrl+R)

Note: Raise is a “sequence of child” operation (not a window operation) that changes the layout in some manager (container) objects.

Moves the selected instance above the sibling directly above it. Use Raise to change the order of menu items, or items in row column or radio boxes.

Example

For example, assume a menu has three push button children, pushButton1, pushButton2, and pushButton3, from top to bottom:

1. Select the bottom-most button, pushButton3.
2. Select Raise from the Browser Edit menu to raise pushButton3 above pushButton2 on the menu.

Lower (Ctrl+L)

Note: Raise is a “sequence of child” operation (not a window operation) that changes the layout in some manager (container) objects.

Moves the selected instance below the sibling directly below it. Use Lower to change the order of menu items, or items in row column or radio boxes.

Example

For example, assume that a menu has three push button children, pushButton1, pushButton2, and pushButton3, from top to bottom:

1. Select the top-most button, pushButton1.
2. Select Lower from the Browser Edit menu to lower pushButton1 below pushButton2 on the menu.

Natural Size (Ctrl+Z)

Removes the height and width resources for the selected instance, causing the instance to calculate its own geometry. (In the placement option menu, None is selected for the height and width resources.) For example, if you change the font size of a label, the widget instance will resize to accommodate that change.

Enlarge (Ctrl+W)

Increases the size of the selected instance. Each time you select Enlarge for an instance, it increments in size.

Align

Displays a menu of the following secondary choices:

Note: The following choices, except Alignment Editor, are insensitive if only one object or no object is selected.

Left	Aligns the left edges of all selected objects with the left edge of the left-most selected object.
Right	Aligns the right edges of all selected objects with the right edge of the right-most selected object.
Top	Aligns the top edges of all selected objects with the top edge of the top-most selected object.
Bottom	Aligns the bottom edges of all selected objects with the bottom edge of the bottom-most selected object.
Horizontal Center	Distributes the centers of all selected objects between the center of the left-most selected object and the center of the right-most selected object.
Vertical Center	Distributes the centers of all selected objects between the center of the top-most selected object and the center of the bottom-most selected object.
Alignment Editor	Displays the Alignment Editor, which allows you to align and distribute multiply-selected objects.

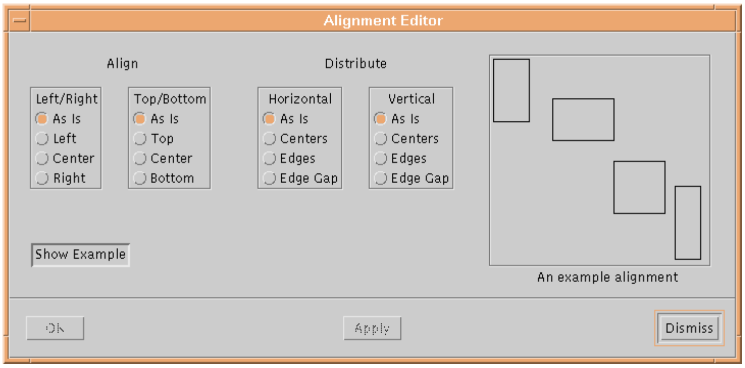


Figure 17. Alignment Editor

Note: OK and Apply are insensitive if one object or no object is selected.

Align

The following table describes how to align objects:

Note: As Is does not align.

Left/Right	Part of Selected Object	Aligned With
Left	Left edge	Left edge of left-most selected widget.
Center	Horizontal center	Midpoint between horizontal centers of left-most and right-most selected widgets.
Right	Right edge	Right edge of right-most selected widget.
Top/Bottom		
Top	Top edge	Top edge of top-most selected widget.
Center	Vertical center	Midpoint between vertical centers of top-most and bottom-most selected widgets.
Bottom	Bottom edge	Bottom edge of bottom-most selected widget.

Distribute

The following table describes how to distribute objects:

Note: As Is does not distribute.

Horizontal	Distribute	Between	And
Centers	Centers of all selected widgets	Center of left-most selected widget	Center of right-most selected widget
Edges	Edges of all selected widgets	Left edge of left-most selected widget	Right edge of right-most selected widget
Edge Gap	Same as Horizontal Edges, but you specify distribution amount (such as “1 cm”) in the input field that appears when you click on this option. All edges are spaced apart by the amount you specify.		
Vertical	**Distribute**	**Between**	**And**
Centers	Centers of all selected widgets	Center of top-most selected widget	Center of bottom-most selected widget
Edges	Edges of all selected widgets	Top edge of top- most selected widget	Bottom edge of bottom-most selected widget
Edge Gap	Same as Vertical Edges, but you specify distribution amount (such as “1 cm”) in the input field that appears when you click on this option. All edges are spaced apart by the amount you specify.		

Edge Gap

When you select Edge Gap, the Spacing Between Edges text field is displayed. Enter both the number and the unit in this field. The following table lists valid units for defining the edge gap:

Unit	Acceptable Abbreviations
pixel	“pix”, “pixel”, or “pixels”
inches	“in”, “inch”, or “inches”
centimeters	“cm”, “centimeter”, or “centimeters”
millimeters	“mm”, “millimeter”, or “millimeters”
points	“pt”, “point”, or “points”
font units	“fu”, “font_unit”, or “font_units”

If you do not specify a unit for Edge Gap, the default unit is pixels. For example:

```
“5 cm” = 5 centimeters

“10 inches” = 10 inches

“40” = 40 pixels
```

When you click on an option, the sample alignment on the Alignment Editor shows you what the option does. The sample alignment always has the same four widget outlines.

Note: This is not meant to be an exact representation of your interface.

Show Example

The Show Example button toggles the display of the sample alignment on and off.

Make/Unmake Class (Ctrl I/Q)

Note: This dialog is automatically displayed when you click on some objects (VkWindow or VkSimpleWindow in ViewKit and when you click on Dialog) from the Palette.

Make Class

Displays a dialog that prompts you for a class name.

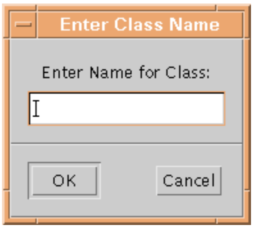


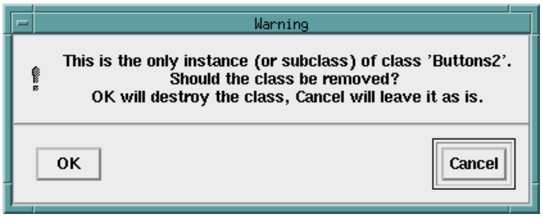
Figure 18. Make Class Dialog

When you enter a name and press return or click the OK button, Builder Xcessory creates a new class comprised of the currently selected object and all of its descendants. The class is represented on the Browser as a single element, with the Class Instance Icon to the left of the class instance name.

Note: This option is insensitive if you select multiple objects.

Unmake Class

Unmake Class is available when a class instance is selected. Displays a dialog prompt to confirm that you want to unmake the instance of the class. If this is the only instance of the class in your interface, displays a warning that unmaking the instance might destroy the class.



This is the only instance (or subclass) of class 'Buttons2'.
Should the class be removed?
OK will destroy the class, Cancel will leave it as is.

Figure 19. Unmake Class Warning Dialog

Make/Unmake Subclass (Ctrl+Y/Q)

Note: Only available when the Browser is in Classes View, and a class is selected.

Make Subclass

Displays a dialog that prompts you for a class name:

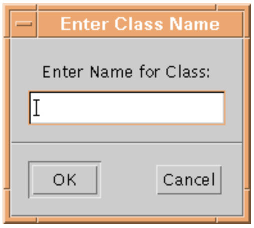


Figure 20. Make Class Dialog

When you enter a name and press return or click the OK button, Builder Xcessory creates a new class comprised of the currently-selected class. The subclass is represented on the Browser as a single element, with the Class Icon to the left of the subclass name. The class from which the subclass was made is displayed as a single element to the right of the subclass on the Browser. A colon appears before the class name, indicating that it has been subclassed.

Adding widgets/class instances to a subclass

When you create subclasses, designate one widget instance on the superclass as a receptor (see the next section, *"Make/Unmake Receptor (Ctrl+T)"*). You can then add widget and/or class instances to the subclass. Any instances you add to the subclass appears on the Browser as children of the element representing the class from which the subclass was created.

Unmake Subclass

Available when the subclass is selected. Unmakes the subclass.

Make/Unmake Receptor (Ctrl+T)

Make Receptor

Designates one container widget instance within a class as the receptor of that class. Any subclass of this class can accept children parented to this receptor widget. When you add a widget to the subclass, the new widget is placed automatically as the child of the receptor, regardless of where you place the widget.

Designating a container widget as the receptor

To designate a container widget instance as that class's receptor:

1. Set the Browser to Classes (instead of Instances) view. The object hierarchy of that class is now displayed in the Browser.
2. Select the container that you want to designate as the receptor.
3. Select Make Receptor from the Browser Edit menu. A square-within-a-square icon indicates which container widget instance has been designated as the receptor.

Unmake Receptor

When you select a receptor widget, this menu option changes to Unmake Receptor. If you then select Unmake Receptor, the selected widget will no longer be the receptor widget for this class. If any subclasses of this class have used this receptor widget (for example, if you have added children to this widget), you cannot unmake the widget.

Note: A class may contain only one receptor.

View Menu

Select View from the Browser menu bar to display the following menu:

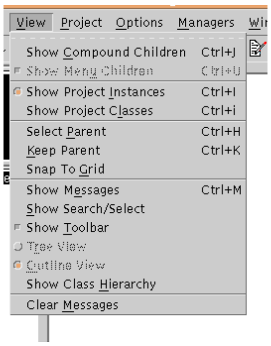


Figure 21. View Menu

The following sections describe each option.

Show Compound Children (Ctrl+J)

Allows you to edit certain resource values of the compound children of compound widgets. Compound children are sub-widgets of an automatically created widget. (For example, if you create a file selector box, it automatically creates buttons.) You can alter the appearance or behavior of the child widgets that comprise a compound widget.

Compound widget view

Selecting Show Compound Children changes the view of any compound widget in the Browser. The Browser displays the compound children (usually hidden) as the children of the compound widget. An icon is displayed next to the children to identify them as special children.

Updating the Resource Editor

You can then select a compound child as you would any other widget, and update the Resource Editor. The Resource Editor displays a subset of the compound child's resources.

Editing resources

Although most resources of a compound child are not available because they are overridden by the containing compound widget, you can edit any displayed resources. Once a compound child's resources are displayed, you can use the Resource Editor to can change resources, hide/show the object, and so forth. Refer to *Chapter 3—Resource Editor* for more detailed information about the Resource Editor.

Note: You cannot use the Browser to cut, copy, or paste a compound child.

Show Menu Children (Ctrl+U)

Allows you to access the contents of menus directly on your interface.

Note: Show Menu Children is insensitive for any currently selected widget other than a cascade button, option menu, or popup menu.

Setting the Show Menu Children toggle for a cascade button or menu displays the associated menu. You can then manipulate the children as you manipulate a typical interface object, for example, using drag and drop operations. Unsetting the Show Menu Children toggle for a given menu hides all of its descendants, and automatically unsets Show Menu Children for any descendant menus.

Show Project Instances(Ctrl+I)

Allows you to enter the Instance View, and display the object instance heirarchy of your interface.

Show Project Classes(Ctrl+i)

Allows you to enter the Class View, and display the object instance heirarchy of all the classes in your interface

Select Parent (Ctrl+H)

Selects the parent of the currently-selected instance and makes the parent the currently-selected instance. Select Parent is especially helpful when a parent is not easily selectable on an interface.

Keep Parent (Ctrl+K)

Forces any widget you subsequently create automatically to become the child of the currently selected widget (saving time in constructing the interface.

Note: Widgets that cannot be legal children of the currently selected widget are desensitized on the Palette. For example, a push button cannot have children. If a push button is the selected widget, and Keep Parent is set, the entire Palette is desensitized.

While Keep Parent is set, you select a new parent widget by clicking on the widget or its instance name. You can create a child of the current parent by double-clicking on any legal object in the Palette.

Snap To Grid

Forces each widget that you subsequently resize, move, or create to align itself with the underlying placement grid. The placement grid is invisible, and you can adjust the grid by selecting User Preferences from the Options menu. Snap To Grid remains set until you unset it. Setting or unsetting Snap To Grid does not move any existing widgets.

Show Messages (Ctrl+M)

Toggles the display of the Browser Message window. If Builder Xcessory sends a warning or error when the window is hidden, the Message window is restored and the message displayed. Informational messages, such as confirmations of file generation, do not automatically restore the Message window.

Click the Hide push button to the left of the Message window to hide the Browser Message window.

Show Search/Select

Displays the Select input field. The Select input field is displayed at the bottom of the Browser.

You can search on either a an Instance of an object (an instance name, such as “pushbutton”) or a Class Instance of an object (a class name, such as “XmPushButton”). To search:

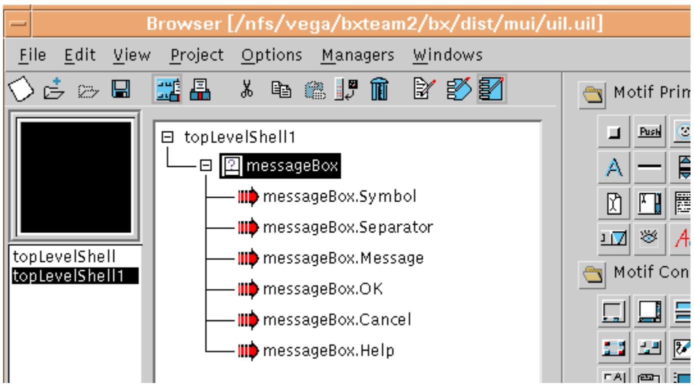
1. Choose either Instance or Class from the option menu next to Select.
2. Enter the name in the Select input field (such as “pushbutton”).

   Every instance that contains that name (such as “pushbutton”, “pushbutton1”, “pushbutton2”, etc.) is selected.

Alternatively, to search for a specific instance that has not been renamed, search on the instance or class name, and use the arrow buttons to walk through one instance at a time.

Show Toolbar

Displays the Browser Toolbar. The Browser Toolbar is displayed immediately under the Browser menu bar. The Browser toolbar is shown by default.



topLevelShell
topLevelShell1

topLevelShell1
messageBox
messageBox.Symbol
messageBox.Separator
messageBox.Message
messageBox.OK
messageBox.Cancel
messageBox.Help

Figure 22. Toolbar

Adding menu items to the Toolbar

To add menu items to the Toolbar, hold down the Shift key and select the item from its menu. For example, to add Raise to the Toolbar:

1. Confirm that the Toolbar is being displayed. If not, set Show Toolbar on the Browser View menu.
2. Hold down the Shift key and select Raise from the Browser Edit menu.

Certain menu items, such as the Windows menu entries, which are automatically entered, cannot be added to the Toolbar. A warning dialog is displayed if you attempt to do so.

Removing menu items from the Toolbar

To remove an item from Toolbar, hold down the Shift key and select either the appropriate icon on the Toolbar or the item on the menu. For example, to remove Raise from the Toolbar, hold the Shift key and select Raise on the Toolbar.

Note: Unselecting Show Toolbar hides the Browser Toolbar.

Show Class Hierarchy

Displays a window that shows the superclass-subclass hierarchy you created in your interface. This window provides an easy method for viewing the organization of your classes.

Clear Messages

Clears any messages currently displayed on the Browser Message window. If Builder Xcessory sends a warning or error when the message window is hidden, the Message window is restored and the message displayed. Informational messages, such as confirmations of file generation, do not automatically restore the Message window.

Note: Click the Hide push button to the left of the Message window to hide the Browser Message window.

Project Menu

The Browser Project menu allows you to construct your interface in Build Mode, test look and feel in Play Mode, run and debug generated code in Debug Mode using your environment's debugger, launch a build of the application you are building in Builder Xcessory, check files in and out of your environment's source code control system, and edit files.

Select Project from the Browser menu bar to display the following menu:

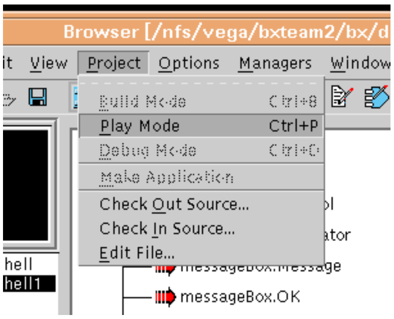


Figure 23. Project Menu

Build Mode (Ctrl+B)

Note: Default mode of Builder Xcessory.

Allows you to use the Palette, Browser, and Resource Editor to create interface objects and edit their resource values.

Play Mode (Ctrl+P)

Allows interface objects to appear as if you compiled and linked your interface, without connected callback structures. All widgets behave normally, and menus, accelerators, mnemonics, and geometry management function as they will in the completed application. The Palette and Resource Editor are withdrawn. Pre-defined callbacks are invoked while Builder Xcessory is in Play Mode. Other callbacks display a message in the Browser Message Window.

If you change resource values while in Play Mode, these values are not necessarily retained when you return to Build Mode (thus allowing you to experiment without changing the values in your interface).

Note: Geometry resource changes are reflected in the interface when you return to Build Mode.

All resource values listed in the Resource Editor (except those for which Placement is set to None) will be Saved or Generated.

Debug Mode (Ctrl+D)

Allows you to launch a build-and-debug session with your environment's debugger. If your environment's build tool is not an integral part of the debugging tool, selecting Debug Mode first launches a build, waits for the build to complete successfully, and loads the debugger with the just compiled application.

Builder Xcessory can make use of the following debuggers.

Development Environment	Debugger	Version Number	Platform
Any environment	CodeCenter	4.x	Solaris, and HP-UX
Any environment	ObjectCenter	2.x	
Sun WorkShop	debugger	6.0	Solaris
SGI Developer Magic	Debugger	2.3 and higher	IRIX

Builder Xcessory actions in Debug Mode

When you select Debug Mode, Builder Xcessory performs the following operations:

- Generates code for your interface.

  If you changed your interface since the last code generation, Builder Xcessory automatically regenerates your source code depending on your currently selected language.

- Builds the interface code, as necessary.

  Builder Xcessory causes the source code to be compiled. If your build tool is an integral part of you debugger, the build is performed by the debugger. Otherwise, the separate tool is used.

- Starts your debugger, as necessary.

  Upon successful compilation, Builder Xcessory starts your debugger (if not already running) using the start command on the Tools Customization dialog.

  Note: You can edit the start command on the Debugger & Build tab of the Tools Customization dialog. Refer to *"Debugger & Build Manager"* on page 103.

  When your debugger starts up, the Builder Xcessory main windows are withdrawn, although you may still make selections from the Browser menus.

- Connects to your debugger.

  You can manage or modify the connection between Builder Xcessory and your debugger on the Debugger & Build tab of the Tools Customization dialog. Refer to *"Debugger & Build Manager"* on page 103.

- Unmaps Builder Xcessory product windows.
- Passes control to your debugger, which runs your interface.

Run, test, and debug your interface. Return to Builder Xcessory to continue development by selecting Build Mode or Play Mode from the Browser Project menu. If you return to Builder Xcessory while your debugger is running your program, Builder Xcessory sends a reset command which might interrupt program run without ending the session.

Note: You can set this reset command for the CenterLine debuggers.

Returning to Builder Xcessory

When you return to Builder Xcessory in either Play Mode or Build Mode, the session remains active. At any time, you can automatically load your interface program back into the debugger/interpreter by selecting Debug Mode from the Browser Project menu.

Changing interfaces

If you are using CodeCenter/ObjectCenter with one interface already loaded, and you want to load an entirely different interface from Builder Xcessory, follow these steps:

1. On CodeCenter/ObjectCenter's command line, enter the following

   ```
   unload user
   ```

2. Select Debug Mode from the Browser Project menu to load and run the new interface.

Note: You must select the Use CenterLine Tools debugger option on the Debugger & Build tab of the Tools Customization dialog. See *"Debugger & Build Manager"* on page 103.

If you initiate a session from Builder Xcessory and then Exit the Builder Xcessory, the session remains running.

Exiting CodeCenter/ ObjectCenter

To exit CodeCenter/ObjectCenter, enter:

```
quit
```

at the CodeCenter/ObjectCenter command line.

Make Application

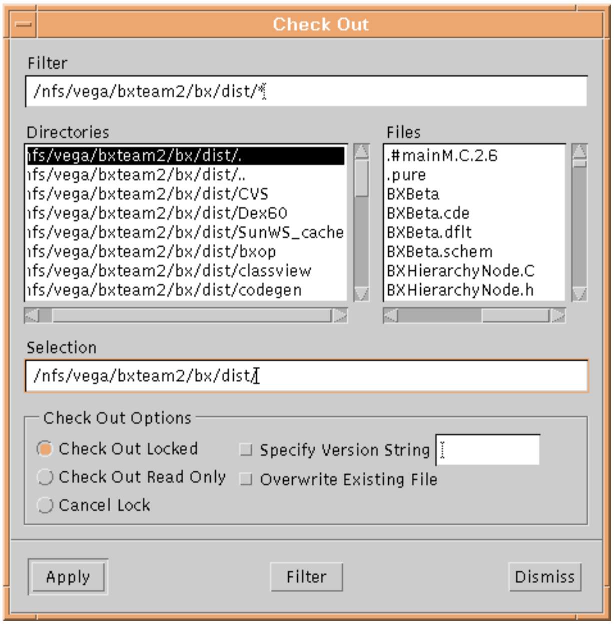
Launches a build of the application for which you are constructing an interface. Builder Xcessory generates code for the application, and causes your environment's build tool to build the application, as shown in the following table:

Platform	Build Tool	Version	Platform
Sun WorkShop	MakeTool	6.0	Solaris
Developer Magic	Build View	2.3 and higher	IRIX
Any Environment	Code/ObjectCenter	4.x and 2.x	Solaris, and HP-UX

Note: The Browser message window displays reports of the build status.

Check Out Source

Displays the Check Out dialog.



Filter: /nfs/vega/bxteam2/bx/dist/*

Directories:
```
nfs/vega/bxteam2/bx/dist/.
nfs/vega/bxteam2/bx/dist/..
nfs/vega/bxteam2/bx/dist/CVS
nfs/vega/bxteam2/bx/dist/Dex60
nfs/vega/bxteam2/bx/dist/SunWS_cache
nfs/vega/bxteam2/bx/dist/bxop
nfs/vega/bxteam2/bx/dist/classview
nfs/vega/bxteam2/bx/dist/codegen
```

Files:
```
.#mainM.C.2.6
.pure
BXBeta
BXBeta.cde
BXBeta.dflt
BXBeta.schem
BXHierarchyNode.C
BXHierarchyNode.h
```

Selection: /nfs/vega/bxteam2/bx/dist/

Check Out Options:
- [✓ Check Out Locked]
- Check Out Read Only
- Cancel Lock
- ☐ Specify Version String
- ☐ Overwrite Existing File

Figure 24. Check Out Dialog

Allows you to check out any file from your environment's source code control system. The default directories and files displayed at the top of the dialog are those in the current working directory. To select a directory and display its files, double-click on the directory name. To select a file, click on the file name.

Selection

The Selection input box displays the currently selected file.

Changing the displayed file

You can change the displayed file by selecting a new file, or by performing the following steps:

1. Click in the input box.
2. Delete its contents.
3. Enter the full pathname of the file you want to check out.

Check Out Options

Use the Check Out Options to determine how the file will be checked out:

- Check Out Locked

  Allows you to lock the file. Another user cannot check out the file until you check it back in.

- Check Out Read Only

  Allows you to check out a read-only version of the file. This checks out the file without locking it.

- Cancel Lock

  Removes a lock. Use this option if you locked a file and no longer need to keep the lock.

- Specify Version String

  Allows you to request a specific version number of the file, if more than one version exists. Enter the version number in the input box.

- Overwrite Existing File

  Allows you to overwrite a read-write version of a currently checked out file.

Note: Before checking out a file, you must select your source code control system. See *“Tools Preferences”* on page 98 for more information about selecting a system.

Check In Source

Displays the Check In dialog:

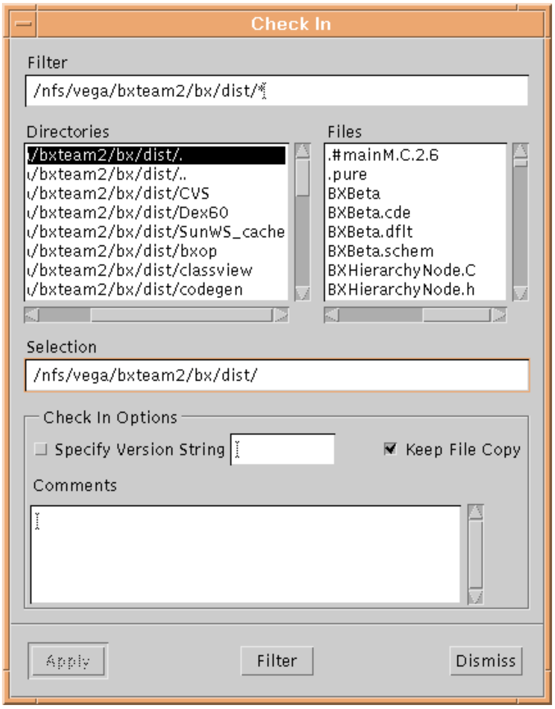


Figure 25. Check In Dialog

Allows you to check a file back into your environment’s source code control system. The default directories and files displayed at the top of the dialog are those in the current working directory.

To select a directory and display its files, double-check on the directory name. To select a file, click on the file name.

Selection

The Selection input box displays the currently selected file.

Changing the displayed file

You can change the displayed file by selecting a new file, or by performing the following steps:

1. Click in the input box.
2. Delete its contents.
3. Enter the full pathname of the file you want to check in.

Check In Options

Use the Check In Options to determine how the file will be checked in.

- Specify Version String

  Allows you to assign a new version number or name to the file, if more than one version exists. Enter the version number in the input box.

- Keep File Copy

  Allows you to maintain a read-only copy of the file you are checking back into the source code control system.

- Comments

  Add comments describing your modifications.

Note: Before checking a file in, select your source code control system. See *"Tools Preferences"* on page 98.

Edit File

Note: You must select an editor from the Tools Customization dialog before using this option. See *“Tools Preferences”* on page 98 for more information about selecting editors.

Allows you to select a file to edit. Select Edit file to display the Edit File file selection dialog:

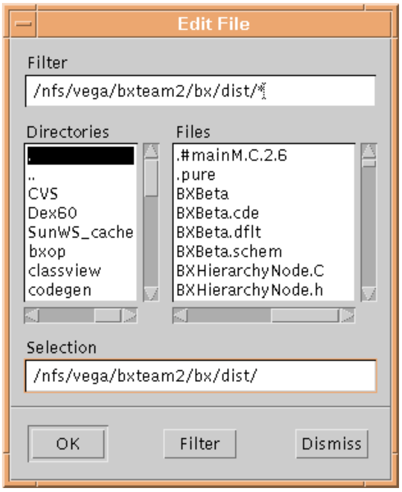


Figure 26. Edit File Dialog

Filter field By default, the Filter field of the File Selection dialog contains the name of the directory from which you are running Builder Xcessory.

Files box The Files box lists the files in the directory which match the file filter, set to “*” by default.

Directories box To display the contents of a different directory, enter the directory name in the Filter field and click the Filter button. You can specify match strings using regular expressions. For example, `*.uil` lists only files ending in the characters `.uil`. Click OK to open the specified file or Cancel to remove the File Selection dialog.

Selection field To select a file, click on the file name in the File box, or enter the full pathname in the Selection field.

Options Menu

The Browser Options menu allows you to change the default language, modify the default language settings, set various user preferences, change information relating to the makefile integration with other tools, internationalization, and the importing of GIL files.

Choose Options from the Browser menu bar to display the following menu:

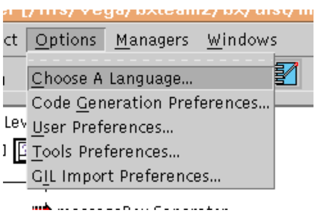


Figure 27. Options Menu

Choose A Language

Displays the Language dialog:

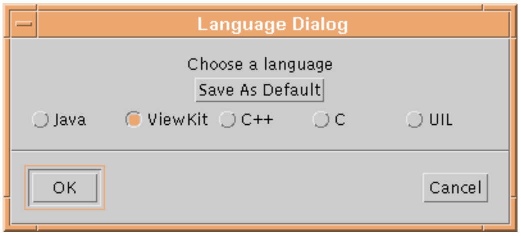
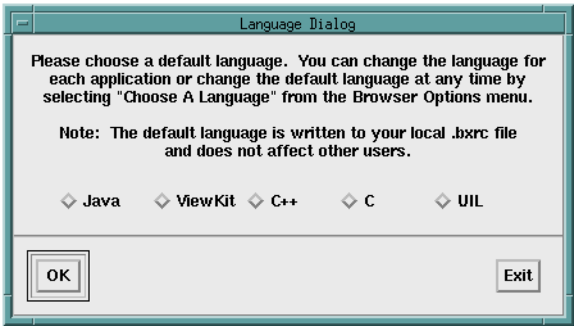


Figure 28. Language Dialog

You can select the language for each application or change the default language at any time during your Builder Xcessory session.

Save As Default Saves the selected language as the default language. The language is saved to the `.bxrc6` file and becomes the default language when you perform a New operation or when you restart Builder Xcessory.

Note: The following Language Dialog is displayed at startup during your first Builder Xcessory session only.


Language Dialog

Please choose a default language. You can change the language for each application or change the default language at any time by selecting "Choose A Language" from the Browser Options menu.

Note: The default language is written to your local .bxrc file and does not affect other users.

Figure 29. Language Dialog at Startup

Code Generation Preferences

Displays the Generation Preferences dialog for the currently specified language. For each tab panel, you can choose the settings for the current project, or save the settings as default values (Save As Default) that are written to your `.bxrc6` file. The default settings appear when you perform a New operation or restart Builder Xcessory.

Note: The Code Generation Preferences are saved with your application in the UIL file. When you open a UIL file, the settings change to the settings for that particular UIL file.

The following sections describe the Code Generation Preferences dialog for each available language.

C++ Code Generation Preferences

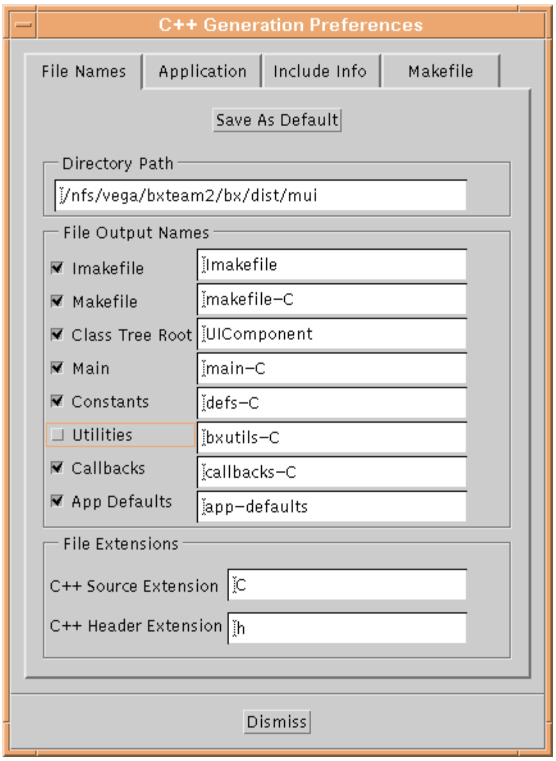
The C++ Generation Preferences dialog is divided into four tabs:

- File Names
- Include Info
- Makefile

The following sections describe the options for each tab panel.

C++ File Names

Click on the File Names tab to display the File Names menu of the C++ Generation Preferences dialog:



Directory Path: /nfs/vega/bxteam2/bx/dist/mui

File Output Names

✓ Imakefile	Imakefile
✓ Makefile	makefile-C
✓ Class Tree Root	UIComponent
✓ Main	main-C
✓ Constants	defs-C
☐ Utilities	bxutils-C
✓ Callbacks	callbacks-C
✓ App Defaults	app-defaults

File Extensions

C++ Source Extension: C

C++ Header Extension: h

Figure 30. File Names Panel on the C++ Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: Displays a warning dialog with the message "This will save the file setting(s) to the BX application resources." Click OK to continue, Cancel to cancel.

Directory Path

The Directory Path field allows you to specify the directory into which generated files will be written.

Note: Changing the directory here changes the directory throughout Builder Xcessory. Builder Xcessory uses only one directory at a time.

File Output Names

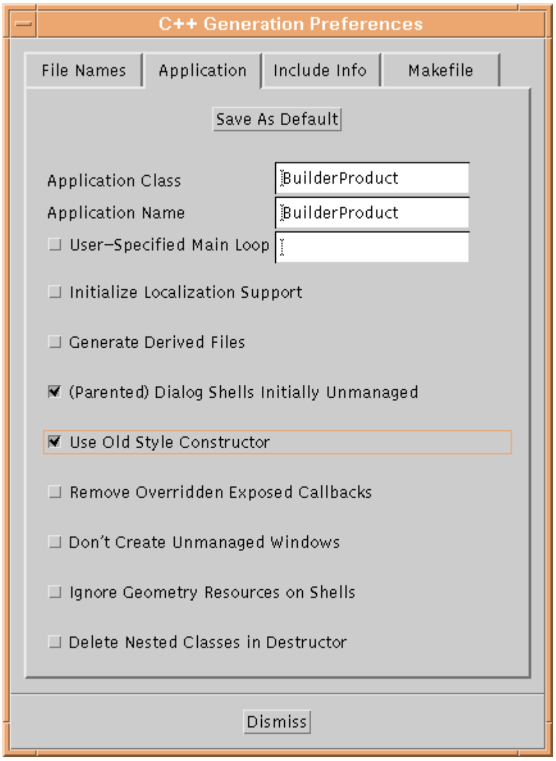
The File Output Names field specifies the following files (defaults are in parentheses):

- Imakefile (`Imakefile`) is the imakefile that specifies C++ imake information.
- Makefile (`makefile-C`) is the C++ makefile.
- Class Tree Root (`UIComponent`) contains the root class from which all classes created in Builder Xcessory are derived.
- Main (`main-C.C)` initializes the Motif toolkit and creates your application's shell widgets and classes. Changing this allows you to use a different base class for your application components. Builder Xcessory creates an unrealized applicationShell and creates all topLevelShell children for all top-level windows. To modify the `main-C.C` file for your application, make your changes in the user code blocks. Any changes made outside the user code blocks will be overwritten in subsequent generations. To write your own main routine, disable subsequent file generation by unsetting the toggle.
- Constants (`defs-C.h`) contains pixmaps, declarations of constants, and global instances of widgets or classes used in the interface.
- Utilities (`bxutils-C.C`) contains convert functions.
- Callbacks (`callbacks-C`) contains any callback information for non-class widgets.
- App Defaults (`app-defaults`) is the file that includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit option `-xrm` on the command line.

File Extensions The File Extensions field specifies the following file extensions:

- C++ Source Extension contains the value appended to the source file name after a dot. The default is `C`.
- C++ Header Extension contains the value appended to the header file after a dot. The default is `h`.

C++ Application Click on the Application tab to display the Application menu of the C++ Generation Preferences dialog:



Application Class: BuilderProduct
Application Name: BuilderProduct
☐ User–Specified Main Loop
☐ Initialize Localization Support
☐ Generate Derived Files
☑ (Parented) Dialog Shells Initially Unmanaged
☑ Use Old Style Constructor
☐ Remove Overridden Exposed Callbacks
☐ Don't Create Unmanaged Windows
☐ Ignore Geometry Resources on Shells
☐ Delete Nested Classes in Destructor

Figure 31. Application Panel on the C++ Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: Displays a warning dialog with the message “This will save the code setting(s) to the BX application resources.” Click OK to continue, Cancel to cancel.

Application Class

The Application Class resource field allows you to enter a class name for the application. Xt uses the Application Class name to find and load application resources, and the application defaults file. When you install your application, the app-default file name should be the same as the application class name.

Application Name

The Application Name resource field allows you to assign a name to your current application. Xt uses the Application Name to resolve precedence of resource specifications.

Toggle options

You can set the following toggle options:

- User-Specified Main Loop specifies a function to call rather than XtAppMainLoop.
- Initialize Localization Support supports whether or not to generate a call to XtSetLanguageProc in the application initialization. Calling this function enables support in X and Motif for non-English locales.
- Generate Derived Files generates an additional subclass for each user-defined class. If set, all application code must be written in the derived file. Default name is “class name”Derived.C.
- (Parented) Dialog Shell Initially Unmanaged toggles to manage Dialog Shells, such as XmDialogShell, in the `main-C.C` file.
- Use Old Style Constructor for backward compatibility. In previous versions of Builder Xcessory, constructors for user-defined classes created the interface, and the parent widget ID was passed as a parameter.
- Remove Overridden Exposed Callbacks removes the class callbacks when overridden. Otherwise both methods will be called.
- Don’t Create Unmanaged Windows delays creation of unmanaged windows.
- Ignore Geometry Resources on Shells does not print shell geometry (x, y, width, height) in the output code.
- Delete Nested Classes in Destructor deletes nested classes created with a New operation inside the create method in the destructor.

C++ Include Info

Click on the Include Info tab to display the Include Info panel of the C++ Generation Preferences dialog:

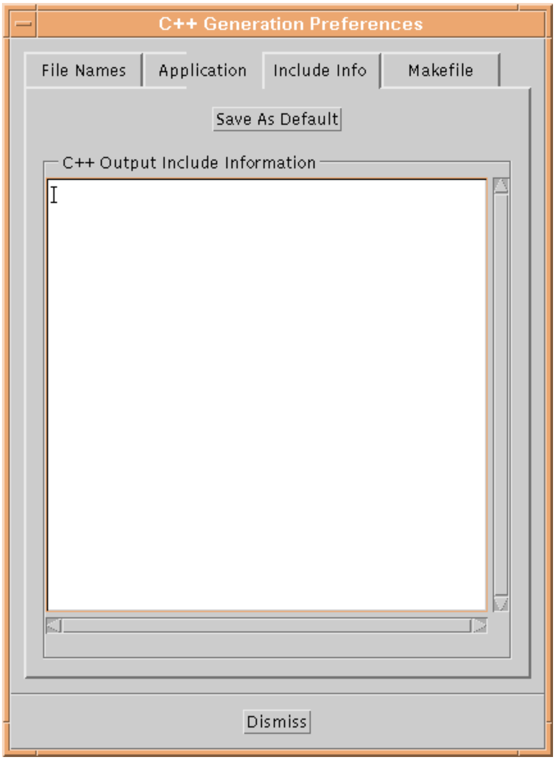
Figure 32. Include Info Panel on C++ Generation Preferences Dialog

Save As Default Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: A warning dialog with the message “This will save the include setting(s) to the BX application resources.” is displayed. Click OK to continue, Cancel to cancel.

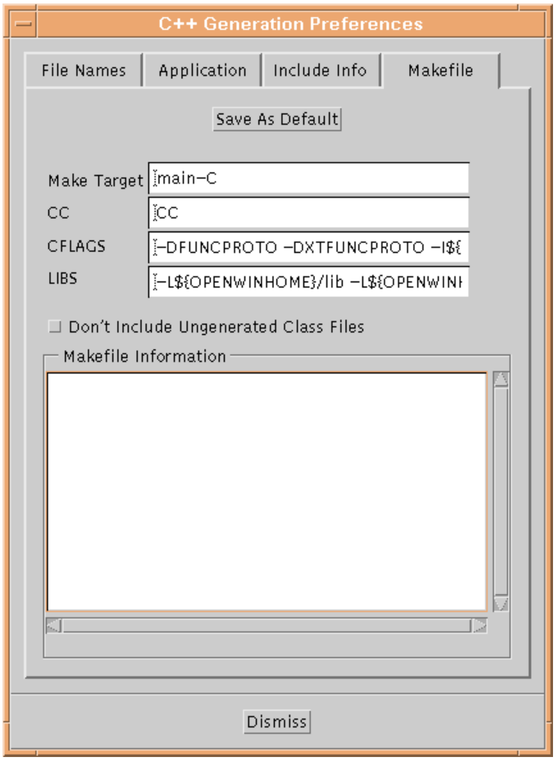
C++ Output Include Information

Any headers or comments you enter in the C++ Output Include Information text field are included at the head of each file generated, under the commented heading `User Supplied Include Files`.



C++ Makefile

Click on the Makefile tab to display the Makefile panel of the C++ Generation Preferences dialog:



Make Target: main-C
CC: CC
CFLAGS: -DFUNCPROTO -DXTFUNCPROTO -I${
LIBS: -L${OPENWINHOME}/lib -L${OPENWINH

☐ Don't Include Ungenerated Class Files

Makefile Information

Figure 33. Makefile Panel on the C++ Generation Preferences Dialog

Save As Default	Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.
	Note: A warning dialog with the message "This will save the make setting(s) to the BX application resources." is displayed. Click OK to continue, Cancel to cancel.
Make Target	Name of the target of the make executed when Make Application is selected from the Browser Project menu (default is `main-C`).
CC	Name of the C++ compiler.
CFLAGS	C++ compiler flags.
LIBS	X, Motif, and any other widget libraries.
Toggle option	Setting the toggle option Don't Include Ungenerated Class Files excludes classes for which you disabled code generation (see *"Generate Class"* on page 149) from the list of source and object files that comprise your application. This is useful if you collected classes into a library, and want to link your application with this library.
Makefile Information	Include additional Makefile comments and information in this text area.

ViewKit Code Generation Preferences

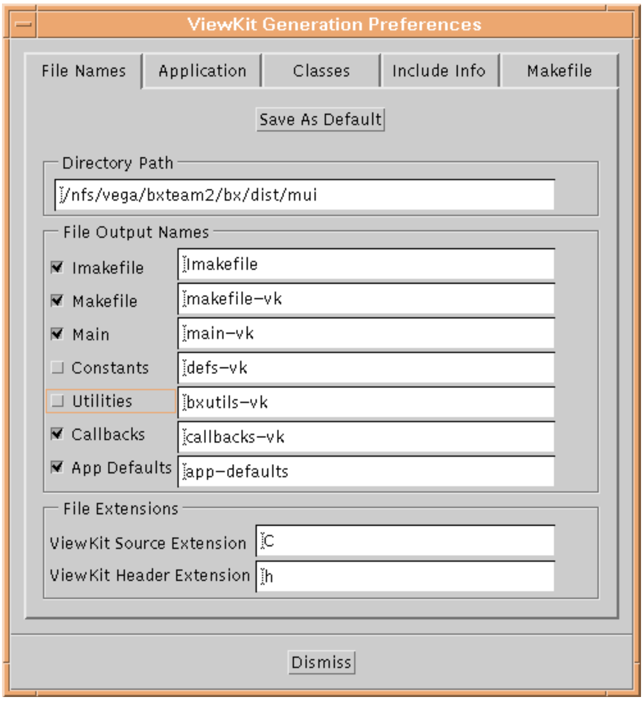
The ViewKit Generation Preferences dialog is divided into the following five tabs:

- File Names
- Application
- Include Info
- Classes
- Makefile Options

The following sections describe the options for each tab panel.

ViewKit File Names

Click on the File Names tab panel to display the File Names menu of the ViewKit Generation Preferences dialog:



Directory Path
/nfs/vega/bxteam2/bx/dist/mui

File Output Names

✓ Imakefile	Imakefile
✓ Makefile	makefile-vk
✓ Main	main-vk
☐ Constants	defs-vk
☐ Utilities	bxutils-vk
✓ Callbacks	callbacks-vk
✓ App Defaults	app-defaults

File Extensions

ViewKit Source Extension	C
ViewKit Header Extension	h

Figure 34. File Names Panel on the ViewKit Generation Preferences Dialog

Save As Default Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: A warning dialog with the message “This will save the file setting(s) to the BX application resources.” is displayed. Click OK to continue, Cancel to cancel.

Directory Path The Directory Path field allows you to specify the directory into which generated files will be written.

Note: Changing the directory here changes the directory throughout Builder Xcessory. Builder Xcessory uses only one directory at a time.

File Output Names The File Output Names field specifies the following files:

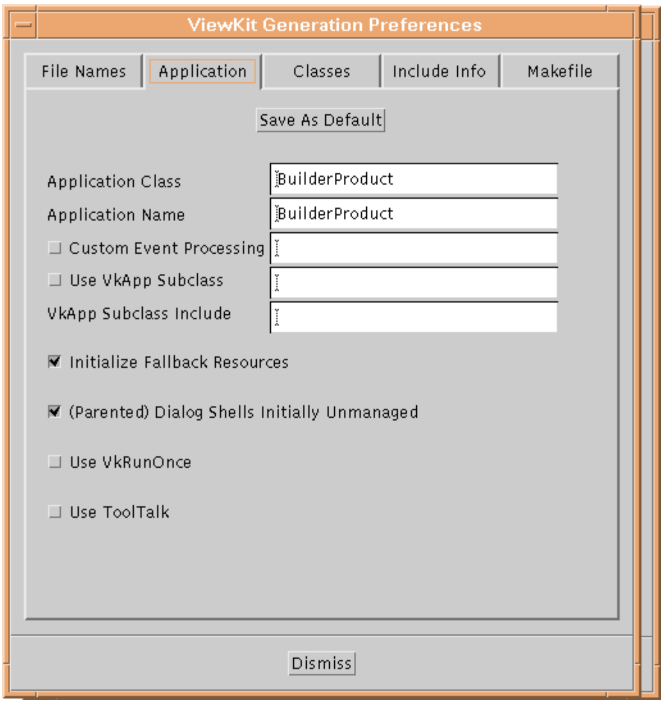
- Imakefile (`Imakefile`)
  Imakefile that specifies ViewKit imake information.
- Makefile (`makefile-vk`)
  ViewKit makefile for the current platform.
- Main (main-vk)
  Initializes the application and instantiates the Window components.
- Constants (`defs-vk`)
  Contains any pixmaps, declarations of constants used in the interface.
- Utilities (`bxutils-vk`)
  Contains convert functions.
- Callbacks (`callbacks-vk`)
  Contains any callback information for non-class objects.
- App Defaults (`app-defaults`)
  File that includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit.

File Extensions The File Extensions field specifies the following file extensions:

- ViewKit Source Extension contains the value appended to the ViewKit source file name after a dot. The default is `C`.
- ViewKit Header Extension contains the value appended to the ViewKit header file after a dot. The default is `h`.

ViewKit Application

Click on the Application tab to display the Application panel of the ViewKit Generation Preferences dialog:



Application Class: BuilderProduct
Application Name: BuilderProduct
☐ Custom Event Processing
☐ Use VkApp Subclass
VkApp Subclass Include
☑ Initialize Fallback Resources
☑ (Parented) Dialog Shells Initially Unmanaged
☐ Use VkRunOnce
☐ Use ToolTalk

Figure 35. Application Panel on the ViewKit Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: A warning dialog with the message “This will save the app setting(s) to the BX application resources.” is displayed. Click OK to continue, Cancel to cancel.

Application Class

The Application Class resource field allows you to enter a class name for the application. Xt uses the Application Class name to find and load application resources, and the application defaults file. When you install your application, the app-default file name should be the same as the application class name.

Application Name

The Application Name resource field allows you to assign a name to your current application. Xt uses the Application Name to resolve precedence of resource specifications.

Toggle options

You can set the toggles for the following options:

- Custom Event Processing allows you to specify that name of a function to pass to VkApp::run(). ViewKit will call this function for every X event received. Builder Xcessory will also generate a stub for this function in `main-vk.C`.
- UseVkApp Subclass allows you to specify a subclass of VkApp to use in `main-vk.C`.

VkApp Subclass Include

Allows you to specify the include file for the VkApp subclass specified when you set the UseVkApp Subclass toggle

Toggle options

You can set the following toggle options:

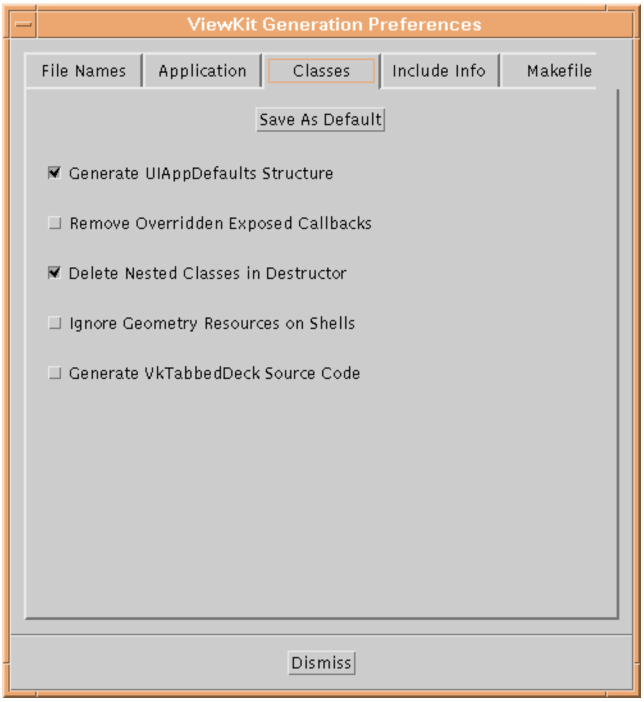
- Remove Overridden Exposed Callbacks removes the class callbacks when overridden. Otherwise both methods will be called.
- Delete Nested Classes in Destructor deletes nested classes created with a New operation inside the create method in the destructor.

Note: The create method must be called to avoid causing an error.

- Initialize Fallback Resources inserts code and a user code block in `main-vk.C` where you can insert you application's fallback errors.
- (Parented) Dialog Shell Initially Unmanaged toggles whether dialogs initially should be hidden regardless of their state in Builder Xcessory.
- Use VkRunOnce allows you to specify that only one single instance of an application can be run on any system at any one time. Also allows you to pass arguments to the running version of the application. Selecting this option will cause an instance of VkRunOne2 to be created in `main-vk.C`.
- UseToolTalk uses VkMsgApp (instead of VkApp) which sets up a ToolTalk session. All windows are subclassed from the VkMsgWindow instead of VkWindow, and VkMsgComponent is used instead of VkComponent. In cases where you specify your own subclasses of VkApp or VkComponent, the generated code assumes they are subclasses of VkMsgApp or VkMsgComponent, respectively.

ViewKit Classes

Click on the ViewKit Classes tab to display the Classes panel of the ViewKit Generation Preferences dialog:



☑ Generate UIAppDefaults Structure
☐ Remove Overridden Exposed Callbacks
☑ Delete Nested Classes in Destructor
☐ Ignore Geometry Resources on Shells
☐ Generate VkTabbedDeck Source Code

Figure 36. Classes Panel on the ViewKit Generation Preferences Dialog

Save As Default Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: Displays a warning dialog with the message “This will save the app setting(s) to the BX application resources.” Click OK to continue, Cancel to cancel.

Toggle options

You can set the following toggle options:

- Generate UIAppDefaults Structure controls whether code for resource defaulting is generated in each class. It is selected by default, largely for backward compatibility. In most cases, the ViewKit supported defaultResources is sufficient and you can deselect this option.
- Remove Overridden Exposed Callbacks removes the class callbacks when overridden. Otherwise both methods will be called.
- Delete Nested Classes in Destructor deletes nested classes created with a New operation inside the create method in the destructor.
- Ignore Geometry Resources on Shells does not print shell geometry (x, y, width, height) in the output code.
- Generate VkTabbedDeck Source Code allows you to choose to have source code for the VkTabbedDeck class generated into you working directory.

Note: Previous ViewKit versions (prior to version 1.3). did not include VkTabbedDeck class.

ViewKit Include Info

Click on the Include Info tab to display the Include Info panel of the ViewKit Generation Preference dialog:

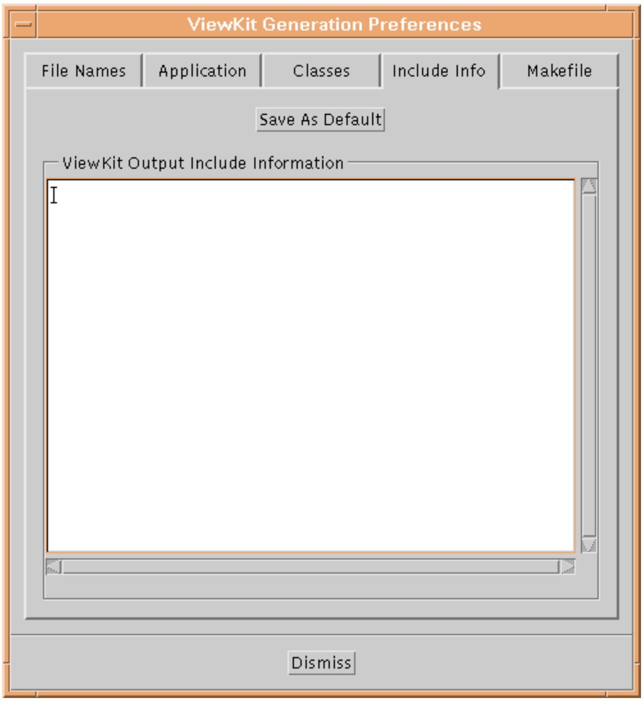


Figure 37. Include Info Panel on the ViewKit Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

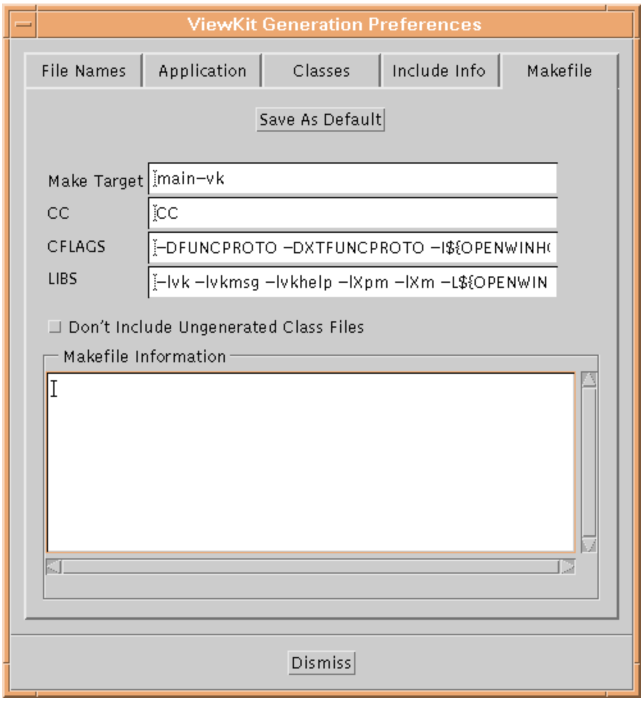
Note: A warning dialog with the message "This will save the make setting(s) to the BX application resources." is displayed. Click OK to continue, Cancel to cancel.

ViewKit Output Include Information

Any headers or comments you enter in the ViewKit Output Include Information text field are included at the head of each file generated, under the commented heading `User Supplied Include Files`.

ViewKit Makefile

Click on the Makefile tab to display the Makefile panel of the ViewKit Generation Preferences dialog:



Make Target: main-vk
CC: CC
CFLAGS: -DFUNCPROTO -DXTFUNCPROTO -I${OPENWINH(
LIBS: -lvk -lvkmsg -lvkhelp -lXpm -lXm -L${OPENWIN

❑ Don't Include Ungenerated Class Files

Makefile Information

Figure 38. Makefile Panel on the ViewKit Generation Preferences Dialog

Save As Default	Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory. **Note:** A warning dialog with the message “This will save the make setting(s) to the BX application resources.” is displayed. Click OK to continue, Cancel to cancel.
Make Target	Name for the command make executed when Make Application is selected from the Browser Project menu. Default is `main-vk`.
CC	Name of the C++ compiler.
CFLAGS	C++ compiler flags.
LIBS	X, Motif, ViewKit and any other widget libraries.
Toggle option	Setting the toggle option Don’t Include Ungenerated Class Files causes all classes for which you chose not to generate code to be excluded from the makefile with the assumption that they are part of a library listed in the LIBS section.
Makefile Information	Include additional Makefile comments and information in this text area.

C Code Generation Preferences

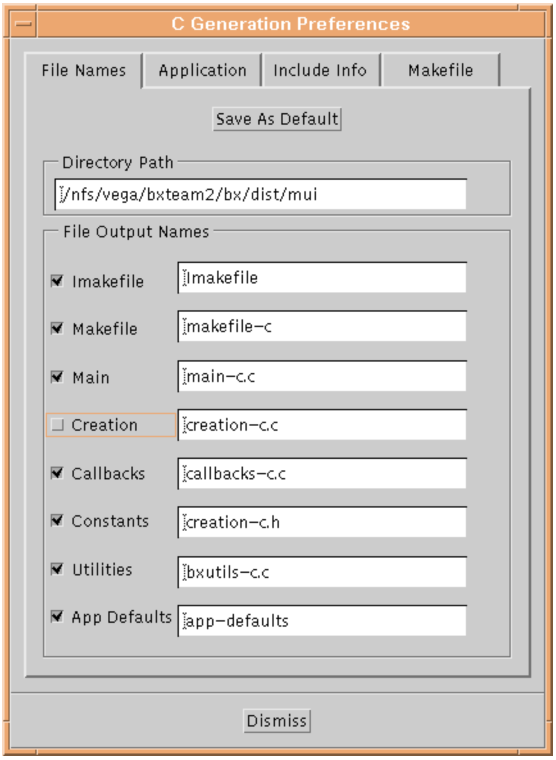
The C Generation Preferences dialog is divided into four tabs:

- File Names
- Application
- Include Info
- Makefile

The following sections describe the options for each tab panel.

C File Names

Click on the File Names tab to display the File Names panel of the C Generation Preferences dialog:



Directory Path: /nfs/vega/bxteam2/bx/dist/mui

File Output Names

✓ Imakefile	Imakefile
✓ Makefile	makefile-c
✓ Main	main-c.c
☐ Creation	creation-c.c
✓ Callbacks	callbacks-c.c
✓ Constants	creation-c.h
✓ Utilities	bxutils-c.c
✓ App Defaults	app-defaults

Figure 39. File Names Panel on the C Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: Displays a warning dialog with the message “This will save the file setting(s) to the BX application resources.” Click OK to continue, Cancel to cancel.

Directory Path

The Directory Path field allows you to specify the directory into which generated files will be written.

Note: Changing the directory here changes the directory throughout Builder Xcessory. Builder Xcessory uses only one directory at a time.

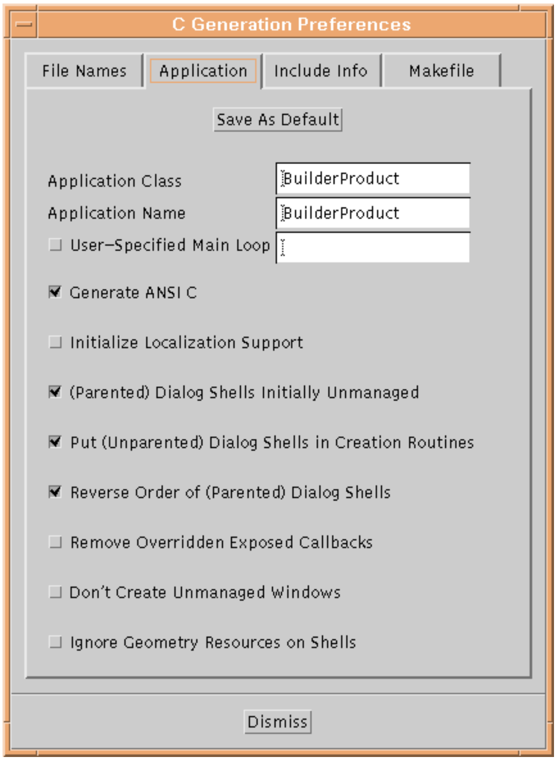
File Output Names

The File Output Names field specifies the following files:

- Imakefile (`Imakefile`)
  Imakefile that specifies C imake information.
- Makefile (`makefile-c`)
  C makefile for the current platform.
- Main (`main-c.c`)
  Initializes the toolkit and calls the Widget Creation routine.
- Creation (`creation-c.c`)
  Contains code to create the widget hierarchy. The functions in this file are called by the Main file.
- Callbacks (`callbacks-c.c`)
  Contains any callback stubs for any callbacks defined in Builder Xcessory. When you write the Callbacks file to a directory with an existing file of the same name, Builder Xcessory scans the existing file for the names of functions and appends callback stubs for undefined callbacks only. Callbacks are never removed from this file or overwritten. Stubs for timers, event handlers, and work procedures are also output in this file.
- Constants (`creation-c.h`)
  Contains any pixmaps, declarations of constants, and global instances of widgets or classes used in the interface.
- Utilities (`bxutil-c.c`)
  Contains convert functions, predefined callback code, and the necessary xpm functions.
- App Defaults (`app-defaults`)
  Includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit.

C Application

Click on the Application tab to display the Application panel of the C Generation Preferences dialog:



Application Class: BuilderProduct
Application Name: BuilderProduct
☐ User–Specified Main Loop
☑ Generate ANSI C
☐ Initialize Localization Support
☑ (Parented) Dialog Shells Initially Unmanaged
☑ Put (Unparented) Dialog Shells in Creation Routines
☑ Reverse Order of (Parented) Dialog Shells
☐ Remove Overridden Exposed Callbacks
☐ Don't Create Unmanaged Windows
☐ Ignore Geometry Resources on Shells

Figure 40. Application Panel on the C Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: A warning dialog with the message “This will save the code setting(s) to the BX application resources.” is displayed. Click OK to continue, Cancel to cancel.

Application Class

The Application Class resource field allows you to enter a class name for the application. Xt uses the Application Class name to find and load application resources, and the application defaults file.

Application Name

The Application Name resource field allows you to assign a name to your current application. Xt uses the Application Name to resolve precedence of resource specifications.

Toggle options

You can set the following options:

- User-Specified Main Loop specifies a function to call in place of the call to XtAppMainLoop.
- Generate ANSI C

  When unset, Builder Xcessory generates standard K&R C code. When set, generates ANSI C. Set by default.
- Initialize Localization Support supports whether or not to generate a call to XtSetLanguageProc in the application initialization. Calling this function enables support in X and Motif for non-English locales.
- (Parented) Dialog Shell Initially Unmanaged toggles to manage Dialog Shells, such as XmDialogShell, in the `main-c.c` file.
- Put (Unparented) Dialog Shells in Creation Routines generates code to create the XmDialogShell in the `creation-c.c` file when a dialog shell that is the child of the root is encountered.
- Reverse Order of (Parented) Dialog Shells reverses the order of Dialog Shells that were not children of the root.
- Remove Overridden Exposed Callbacks removes the class callbacks when overridden. Otherwise both methods will be called.
- Don’t Create Unmanaged Windows delays creation of unmanaged windows.

C Include Info Click on the Include Info tab to display the Include Info panel of the C Generation Preferences dialog:

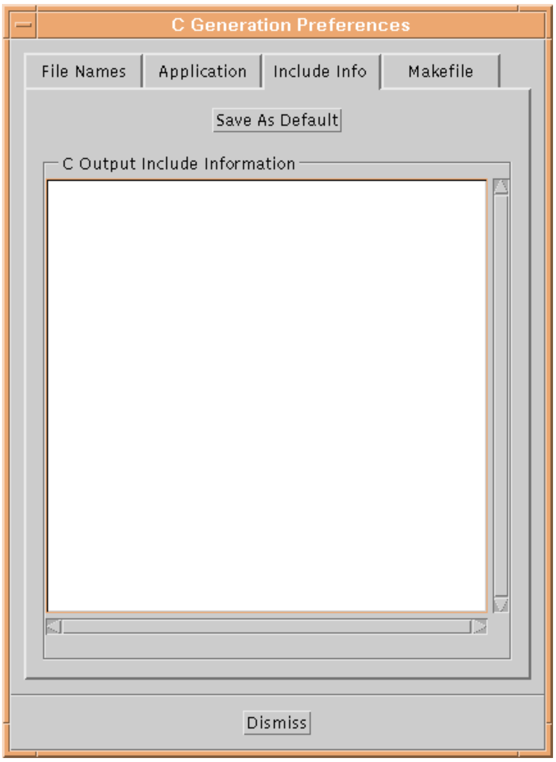


Figure 41. C Include Info Tab on the C Generation Preferences Dialog

Save As Default Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

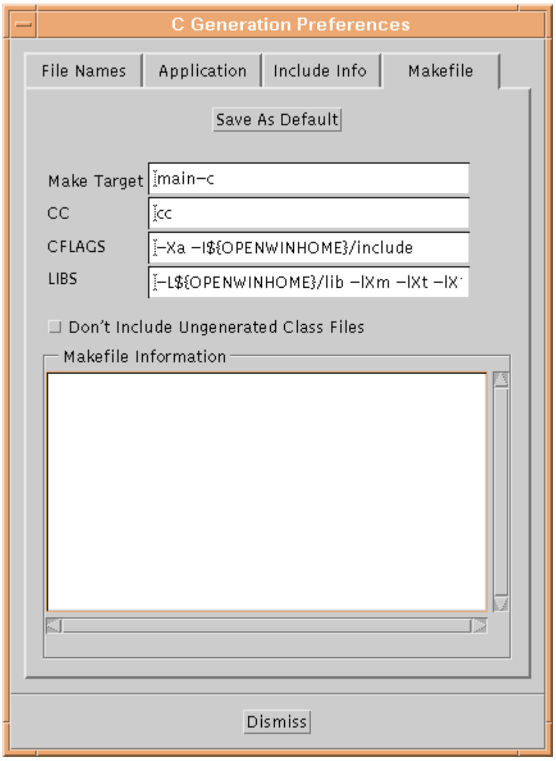
Note: Displays a warning dialog with the message “This will save the include setting(s) to the BX application resources.” Click OK to continue, Cancel to cancel.

C Output Include Information

Any headers or comments you enter in the C Output Include Information text field are included at the head of each file generated, under the commented heading `User Supplied Include Files`.

C Makefile

Click on the Makefile tab to display the Makefile panel of the C Generation Preferences dialog:



Make Target: main-c
CC: cc
CFLAGS: -Xa -I${OPENWINHOME}/include
LIBS: -L${OPENWINHOME}/lib -lXm -lXt -lX

Figure 42. C Makefile Tab on the C Generation Preferences Dialog

Save As Default	Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory. **Note:** Displays a warning dialog with the message “This will save the make setting(s) to the BX application resources.” Click OK to continue, Cancel to cancel.
Make Target	Name of the target of the make executed when Make Application is selected from the Browser Project menu (default is `main-c`).
CC	Name of the C compiler.
CFLAGS	C compiler flags.
LIBS	X, Motif, and any other widget libraries.
Toggle option	Setting the toggle option Don’t Include Ungenerated Class Files excludes classes for which you disabled code generation (see *“Generate Class”* on page 149) from the list of source and object files that comprise your application. This is useful if you collected classes into a library, and want to link your application with this library.
Makefile Information	Include additional Makefile comments and information in this text area.

UIL Code Generation Preferences

The UIL Generation Preferences dialog is divided into four tabs:

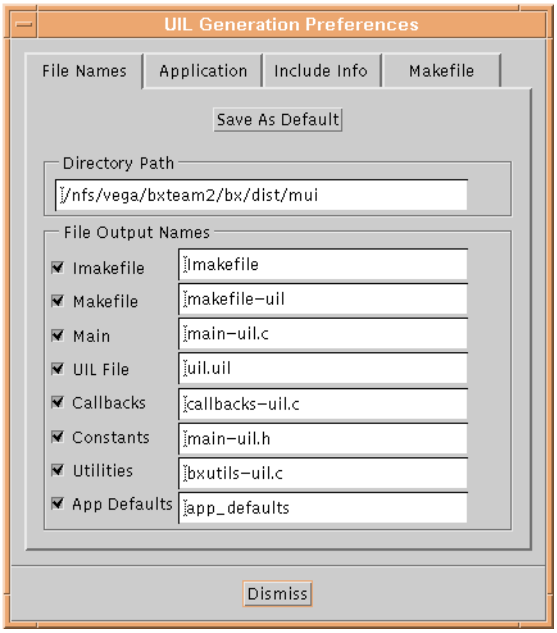
- File Names
- Application
- Include Info
- Makefile Option

Note: When you select UIL as your language, Builder Xcessory generates UIL to implement the interface and C code to implement the application.

The following sections describe the options for each tab panel.

UIL File Names

Click on the File Names tab to display the File Names panel of the UIL Generation Preferences dialog:



Directory Path: /nfs/vega/bxteam2/bx/dist/mui

File Output Names

✓ Imakefile	Imakefile
✓ Makefile	makefile-uil
✓ Main	main-uil.c
✓ UIL File	uil.uil
✓ Callbacks	callbacks-uil.c
✓ Constants	main-uil.h
✓ Utilities	bxutils-uil.c
✓ App Defaults	app_defaults

Figure 43. File Names Menu on the UIL Generation Preferences Dialog

Save As Default

Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: A warning dialog with the message "This will save the file setting(s) to the BX application resources." is displayed. Click OK to continue, Cancel to cancel.

Directory Path

The Directory Path field allows you to specify the directory into which generated files will be written.

Note: Changing the directory here changes the directory throughout Builder Xcessory. Builder Xcessory uses only one directory at a time.

File Output Names

The File Output Names field specifies the following files:

- Imakefile (`Imakefile`)
  Imakefile that specifies both UIL and C imake information.
- Makefile (`makefile-uil`)
  Makefile for the current platform. Has targets for UIL as well as the associated C files.
- Main (`main-uil.c`)
  Initializes the toolkit, initializes Mrm, and makes Mrm calls in order to create widgets. Although this file is intended as a template to enable you to immediately compile the interface output from Builder Xcessory, it will not be adequate for all applications. In particular, Builder Xcessory creates an unrealized applicationShell and creates all topLevelShell children for all top-level windows, as recommended in Asente, Converse, and Swick's *X Window System Toolkit*.

  Note: You can modify this file within the specified user code blocks, but changes made outside these blocks will not be preserved between code generation sessions.

- UIL File (`uil.uil`)

  Contains a description of the widgets. Compiled to a `.uid` file which is then read by the Mrm calls at run-time to create the widgets. Builder Xcessory generates a backup file whenever it overwrites an existing UIL file. Automatic backup is provided for the default filename `uil.uil` or any customized filename. If, for example, you write the renamed UIL file `new_uil.uil`, and a copy of this file already exists, Builder Xcessory generates the backup file `new_uil.uil~`. These UIL files are identical to the files generated when you select Save from the Browser File menu.

- Callbacks (`callbacks-uil.c`)

  Contains callback stubs. When you write out the Callbacks file to a directory with an existing file of the same name, Builder Xcessory scans the existing file for the names of functions and appends callback stubs for undefined callbacks only. Callbacks are never removed from this file and are never overwritten. Stubs for timers, event handlers, and work procedures are also output in this file.

- Constants (`main-uil.h`)

  Contains declarations of constants, and global instances of widgets or classes used in the interface.

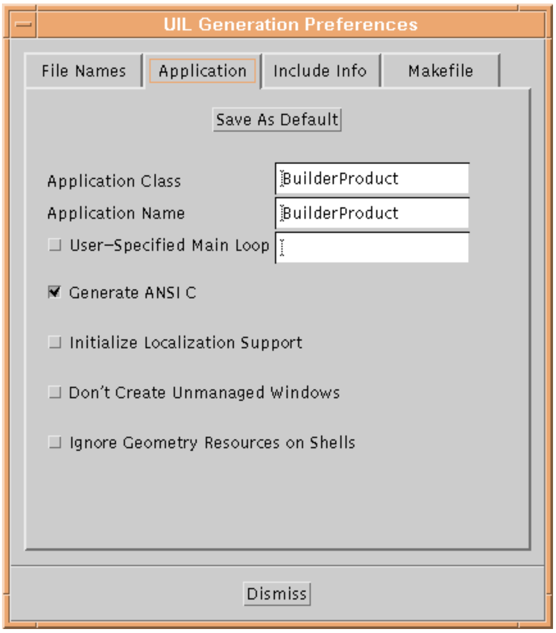
- Utilities (`bxutil-uil.c`)

  Contains convert functions and predefined callback code.

- App Defaults (`app-defaults`)

  File that includes all resources with app-defaults value set to App. The user can change an app-default resource value by editing the app-defaults resource file, overriding the file from a local resource file, or entering the toolkit

UIL Application Click on the Application tab to display the Application panel of the UIL Generation Preferences dialog:



Application Class: BuilderProduct
Application Name: BuilderProduct
☐ User-Specified Main Loop
☑ Generate ANSI C
☐ Initialize Localization Support
☐ Don't Create Unmanaged Windows
☐ Ignore Geometry Resources on Shells

Figure 44. Application Panel on the UIL Generation Preferences Dialog

Save As Default Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: A warning dialog with the message “This will save the app setting(s) to the BX application resources.” is displayed. Click OK to continue, Cancel to cancel.

Application Class The Application Class resource field allows you to enter a class name for the application. Xt uses the Application Class name to find and load application resources, and the application defaults file.

Application Name The Application Name resource field allows you to assign a name to your current application. Xt uses the Application Name to resolve precedence of resource specifications.

Toggle options

You can set the following options:

- User-Specified Main Loop specifies a main loop to replace XtAppMainLoop.
- Generate ANSI C

  When unset, Builder Xcessory generates standard K&R C code. When set, generates ANSI C. Set by default.
- Don’t Create Unmanaged Windows delays creation of unmanaged windows.
- Ignore Geometry Resources on Shells does not print shell geometry (x, y, width, height) in the output code.

UIL Include Info

Click on the Include Info tab to display the Include Info menu of the UIL Generation Preferences dialog:

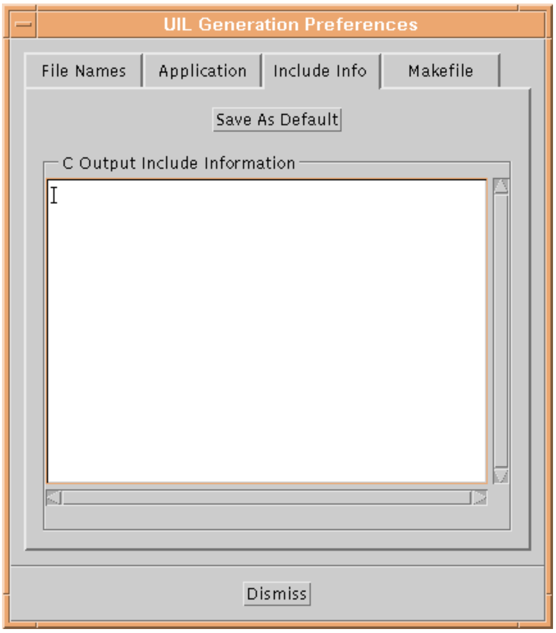
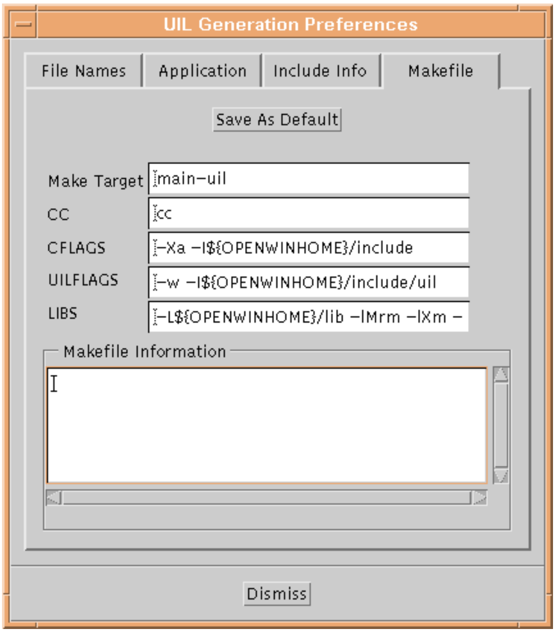


Figure 45. Include Info Panel on the UIL Generation Preferences Dialog

C Output Include Information

Any headers or comments you enter in the C Output Include Information text field are included at the head of each file generated, under the heading `User Supplied Include Files`.

UIL Makefile Click on the Makefile tab to display the Makefile panel of the UIL Generation Preferences dialog:



Make Target: main-uil
CC: cc
CFLAGS: -Xa -I${OPENWINHOME}/include
UILFLAGS: -w -I${OPENWINHOME}/include/uil
LIBS: -L${OPENWINHOME}/lib -lMrm -lXm -

Figure 46. Makefile Panel on the UIL Generation Preferences Dialog

Save As Default	Saves the specified settings on this menu as the default settings. The settings are saved to the `.bxrc6` file and appear when you perform a New operation or when you restart Builder Xcessory.

Note: Displays a warning dialog with the message "This will save the include setting(s) to the BX application resources." Click OK to continue, Cancel to cancel.

Make Target	Make Target (`main-uil`) is the name of the target of the make command executed when you select Make Application from the Browser Project menu.
CC	Name of the C compiler.
CFLAGS	C compiler flags.
LIBS	X, Motif, and any other widget libraries.

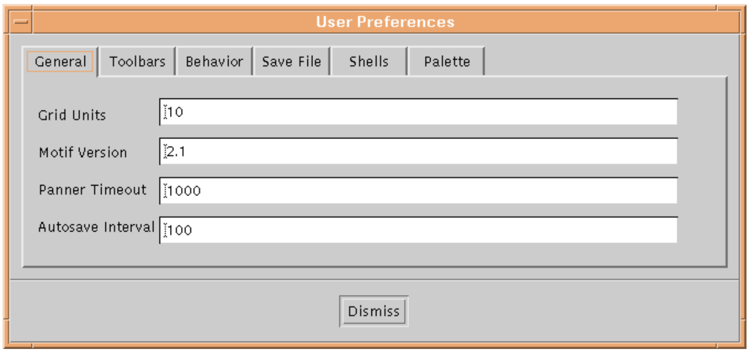
IUser Preferences

Displays the User Preferences dialog. The User Preferences dialog is divided into the following five tabs:

- General
- Toolbars
- Behavior
- Save File
- Shells
- Pallette

The following sections describe the options for each tab panel.

General Click on the General tab of the User Preferences dialog to display the General panel:



Grid Units: 10
Motif Version: 2.1
Panner Timeout: 1000
Autosave Interval: 100

Figure 47. General Panel of the User Preferences Dialog

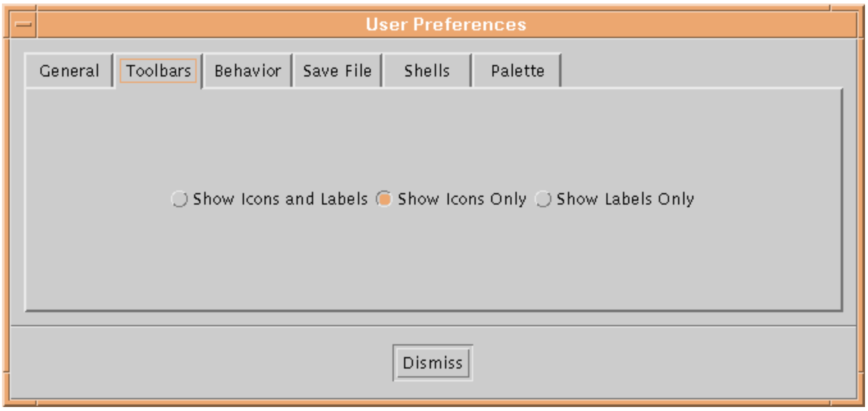
Grid Units Allows you to change the placement grid dimensions. Enter the width of a grid square in pixels in the Grid Units text field. Default value is 10.

Motif Version Allows you to specify the version of Motif. Default value is 2.1.

Panner Timeout Allows you to specify the speed (in milliseconds) of the automatic panning of the Browser display. Default value is 1000.

Autosave Interval Allows you to specify the interval, in button and key actions, between automatic backups of the UIL file. Default value is 100.

Toolbars Click on the Toolbars tab of the User Preferences dialog to display the Toolbars panel:



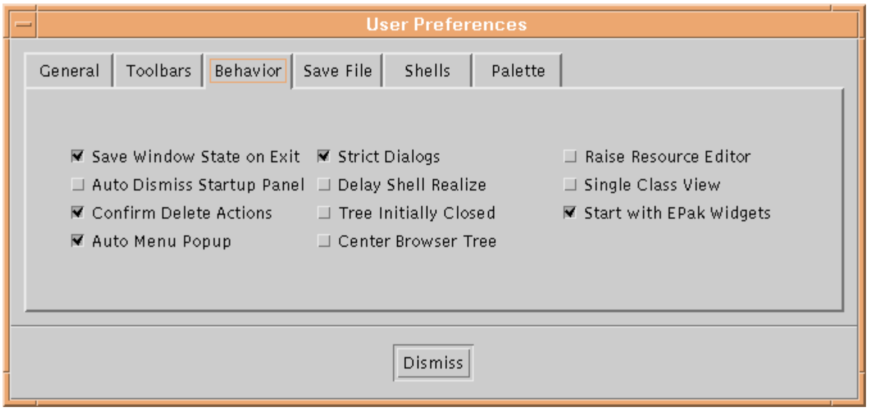
( ) Show Icons and Labels [✓ Show Icons Only] ( ) Show Labels Only

Figure 48. Toolbars Panel of the User Preferences Dialog

Toolbars options Controls the display of both the Browser and the Resource Editor Toolbars. You can set the following toggle options:

- Icons and Labels
- Icons Only
- Labels Only

Behavior Click on the Behavior tab of the User Preferences dialog to display the Behavior panel:



- [✓] Save Window State on Exit
- [ ] Auto Dismiss Startup Panel
- [✓] Confirm Delete Actions
- [✓] Auto Menu Popup
- [✓] Strict Dialogs
- [ ] Delay Shell Realize
- [ ] Tree Initially Closed
- [ ] Center Browser Tree
- [ ] Raise Resource Editor
- [ ] Single Class View
- [✓] Start with EPak Widgets

Figure 49. Behavior Panel of the User Preferences Dialog

Behavior toggle options

You can set the following toggle options:

- Save Window State on Exit

  Saves the size, position, and icon state of Builder Xcessory windows between sessions. When set, the window state of your current Builder Xcessory session will be used the next time you start Builder Xcessory.

- Auto Dismiss Startup Panel

  Dismisses the ICS Window after Builder Xcessory finishes loading. Unset by default.

- Confirm Delete Actions

  When set, all deletions must be confirmed by the user. Set by default.

- Auto Menu Popup

  When set, the children of a menu are displayed on the interface when you click on the menu parent, and when you create a menu. Set by default.

- Strict Dialogs

  According to the OSF/Motif Style Guide, only those objects which inherit from the XmBulletinBoard widget class may be children of an XmDialogShell. When Strict Dialogs is set, Builder Xcessory complies with that recommendation. When Strict Dialogs is unset, Builder Xcessory permits you to create any object as the child of an XmDialog Shell. Set by default.

- Delay Shell Realize

  Unset by default. When set, each top level shell of an interface read into or opened by Builder Xcessory is initially deselected in the Browser list, and the widget tree is not realized until the top level shell is selected. This feature allows you to read or open very large interfaces more efficiently.

- Tree Initially Closed

  When unset, the descendants of an object created along with that object are displayed on the Browser object instance hierarchy. The top level shell or parent's folder defaults to Open. When Tree Initially Closed is set, the descendants of an object created along with that widget are hidden on the Browser object instance hierarchy. (The top level shell or parent's folder defaults to Closed.) Unset by default.

- Center Browser Tree

  When set, the object instance hierarchy display on the Browser is automatically centered around the currently selected object. Unset by default.

- Raise Resource Editor

  When unset, the Resource Editor remains where it is on your display (even iconified) when you double-click on an object to update the Resource Editor. When set, the Resource Editor is raised to the top of your display (and deiconified as necessary) when you double-click on an object to update the Resource Editor. Unset by default.

- Single Class View

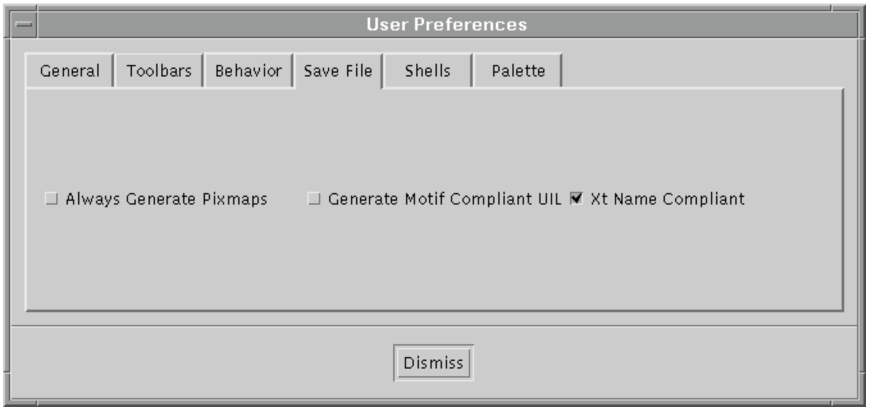
  When set, shows only one class at a time. Clicking on another class in the list hides the current class and shows the new class. Useful for browsing large collections of classes. Unset by default.

- Start with EPak Widgets

  When set, the Palette includes the members of the ICS Motif EnhancementPak on startup. Unset by default in Builder Xcessory. Set by default in BX PRO.

Save File

Click on the Save File tab of the User Preferences dialog to display the Save File panel:



☐ Always Generate Pixmaps ☐ Generate Motif Compliant UIL ☑ Xt Name Compliant

Figure 50. Save File Panel of the User Preferences Dialog

Save File toggle options

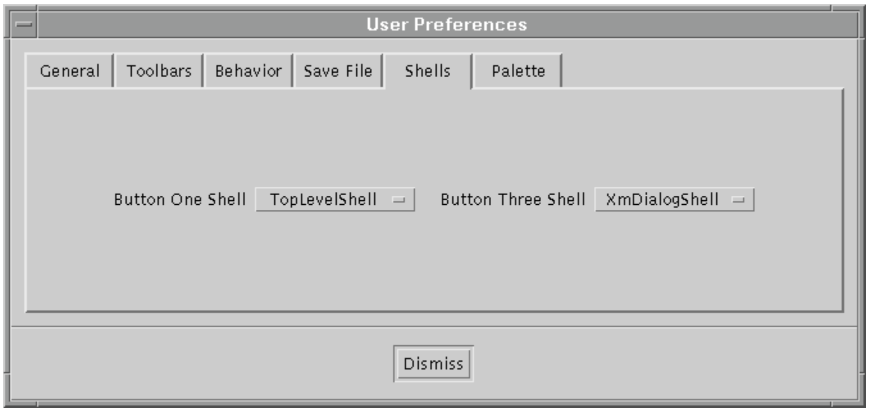
You can set the following toggle options:

- Always Generate Pixmaps generates code for any pixmaps loaded into Builder Xcessory, even if unused. Typically, Builder Xcessory will generate only code for pixmaps used in the application. Unset by default.
- Generate Motif CompliantUIL

  When set, Builder Xcessory outputs user-defined widgets to compile with the standard UIL compiler (without rebuilding). Unset by default.
- Xt Name Compliant

  When unset, allows instance names of widgets to begin with an upper case letter. When set, forces a lowercase letter. Capitalized widget names might cause problems with the proper inheritance of application defaults resource values and conflicts with Xt specifications. Set by default.

Shells Click on the Shells tab of the User Preferences dialog to display the Shells Panel:



Button One Shell TopLevelShell
Button Three Shell XmDialogShell

Figure 51. Shells Panel of the User Preferences Dialog

Button One Shell This menu allows you to select the kind of shell created when you drag and drop an object from the Palette with MB1. The shell options on the scroll menu include the following:

- ApplicationShell
- TopLevelShell
- XmDialogShell
- TransientShell

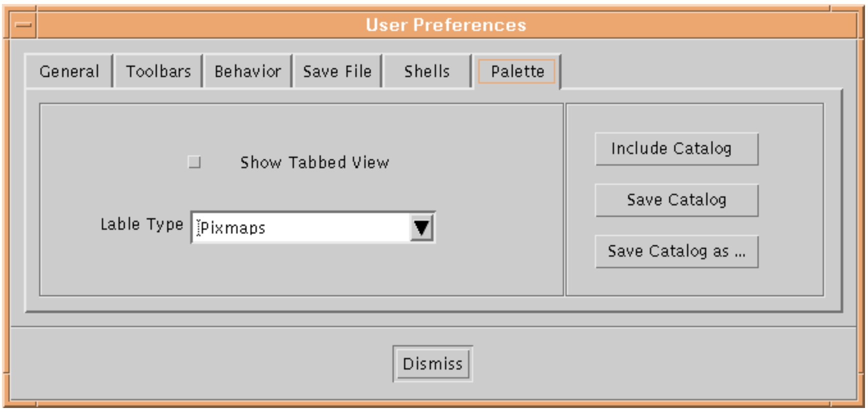
Default value is TopLevelShell.

Button Three Shell This menu allows you to select which kind of shell is created when you drag and drop an object from the Palette with MB3. The shell options on the scroll menu include the following:

- ApplicationShell
- TopLevelShell
- XmDialogShell
- TransientShell

Default value is XmDialogShell.

Pallette Click on the Pallette tab of the User Preferences dialog to display the Pallette Panel:



Show Tabbed View

Lable Type Pixmaps

Figure 52. Pallette Panel of the User Preferences Dialog

Show Tabbed View This toggle button allows you to display the Pallette icons on tabbed or outline format. The default is Outline View. You can click on Show Tabbed View to Toggle the view.

Label Type This combobox allows you to select how the pallette icons are displayed:

- Labels only - Displays the name of each pallette widget, class, or collection. By default, user collections are assigned the unique names Collection000, Collections001, etc. in the order of thier creation
- Pixmaps Only - Displays only the pixmap icon for each pallette widget, class, or collection. User collections are assigned a default pixmap.
- Both - Displays all pixmap icons labeled with their respective widget collection names

Default value is Pixmap Only.

Include Catalog This button openes a Open File Dialog: You may select a catalog to include on your pallette. See “Customizing Builder Xcessory” documentation for details on how to create a catalog.

Save Catalog This button saves the current pallette in the default catalog file. See “Customizing Builder Xcessory” documentation for details on the default catalog file:

Save Catalog As This button saves the current pallette in a user named catalog file. See “Customizing Builder Xcessory” documentation for details on catalog files:

Tools Preferences

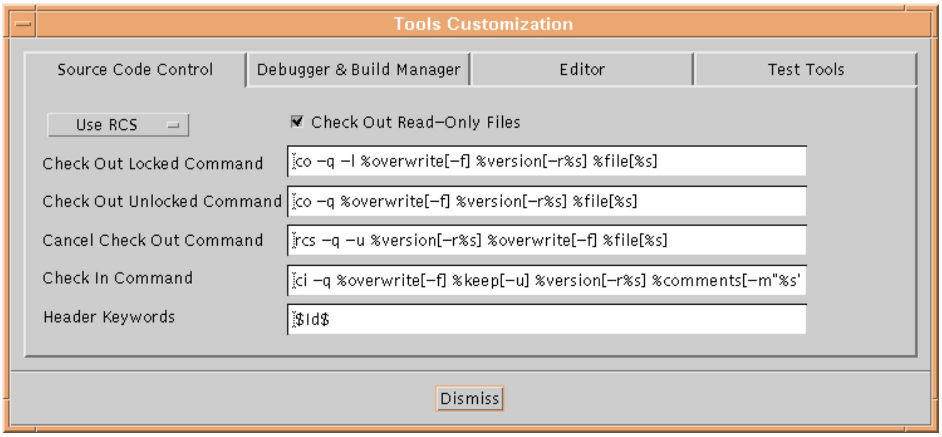
Displays the Tools Customization dialog. The Tools Customization dialog is divided into the following tabs:

- Source Code Control
- Debugger & Build Manager
- Editor
- Test Tools

The following sections describe the options for each tab panel.

Note: Some of the tab options are available only when you configure Builder Xcessory to use a development environment tool.

Source Code Control Click on the Source Code Control tab of the Tools Customization dialog to display the Source Code Control panel:



Use RCS

☑ Check Out Read–Only Files

Check Out Locked Command	co –q –l %overwrite[–f] %version[–r%s] %file[%s]
Check Out Unlocked Command	co –q %overwrite[–f] %version[–r%s] %file[%s]
Cancel Check Out Command	rcs –q –u %version[–r%s] %overwrite[–f] %file[%s]
Check In Command	ci –q %overwrite[–f] %keep[–u] %version[–r%s] %comments[–m"%s'
Header Keywords	Id

Figure 53. Source Code Controls Panel of the Tools Customization Dialog

Allows you to reset the value of several resources related to the following options (available based on the supported environments and platforms).

Select one of the available systems, including your environment’s source code control system. CVS (Concurrent Versions System), RCS (Revision Control System), SCCS (Source Code Control System), and other source code control systems allow you to manage code in a multi-developer environment.

Typically, the program forces you to check out a file to modify it, and prevents other users from modifying the file until it is checked back in. Source code control programs also keep track of the revisions made between subsequently checked-in files, to allow quick reconstruction of a previous version of a file. Consult your source code control documentation for more detailed information.

The source code menu allows you to select your source code control system. If you start Builder Xcessory from within your development environment, select the Use Environment Manager option to use your environment’s system.

Scroll menu options

If your environment does not have a source code control system, or you want to use another system, select one of the following options:

- Unset

  Disables the use of source code control within Builder Xcessory. The Check Out and Check In options in the File menu become insensitive, and Builder Xcessory does not query whether a file should be checked out.

  Note: If your development environment supports integration with its source code control tool, this option might be listed as an option to use your environment’s source code control tool.

- Use CVS

  Fills the text fields below the option menu with the default CVS commands for locking a file, checking out a file, cancelling a check out, and checking in a file. You can edit any of these text fields.

- Use RCS

  Fills the text fields below the option menu with the default RCS commands for locking a file, checking out a file, cancelling a check out, and checking in a file. You can edit any of these text fields.

- Use SCCS

  Fills the text fields below the option menu with the default SCCS commands for checking out and locking a file, checking out a file, cancelling a check out, and checking in a file. You can edit any of these text fields.

- Use ClearCase

  Fills the text fields below the option menu with the default ClearCase commands for checking out and locking a file, checking out a file, cancelling a check out, and checking in a file. You can edit any of these text fields.
- User Specified

  Clears the text fields. You must supply command strings for each field. If you do not enter a command string in a field, its associated action will not be available through the Check Out and Check In dialogs.

Source Code Control text fields

The following text fields are available on the Source Code Control menu (defaults depend on whether you selected CVS, RCS, SCCS, Clear Case or User Specified):

- Check Out Locked Command

  Checks file out from source code control system, makes the file editable, and locks it.
- Check Out Unlocked Command

  Gets a read-only version of the most recent version of the file.
- Cancel Check Out Command

  Cancels a check-out command. Any changes you make to the checked-out file are ignored. The original file remains intact and checked in.
- Check In Command

  Checks the edited file back in.
- Header Keywords

  Revision control for CVS, RCS, SCCS, Clear Case or User Specified. Allows you to insert identification keywords in Builder Xcessory `.uil` file. The default for RCS is `$ID$`. The default for SCCS is `%W% %D% %T%.` The other tools default to blank. Consult the appropriate source code control documentation for more information.

Options syntax in Builder Xcessory

Builder Xcessory executes many of the source code control commands by constructing a command as it would be entered at a shell prompt. The command lines for the various commands use symbolic representations for the various user-specified command line options.

Builder Xcessory substitutes the appropriate value for the representation when it builds the command line. In general, the options syntax is as follows:

```
%option_name[actual_option_flag]
```

Everything within the brackets “[ ]” is substituted for the option name in the command line. Most options may also include “%s” within the brackets “[ ]”, indicating where to insert dynamically determined text, (such as a file name or a line number).

Example

For example, assume a command to check out a file from source code control was "checkout" and it takes the name of the file to check out as an argument. The following is entered:

```
checkout %file[%s]
```

If the filename argument must follow the flag "-f", the command is entered as follows:

```
checkout %file[-f%s]
```

Note: Items outside an option description, that is, not within the brackets "[ ]", are added verbatim in the command line.

Command line option substitutions

The following list describes the command line option substitutions available for source code control commands:

Note: None of these options are required.

- %file

  Indicates the filename for the source code control operation. You can use %s within the brackets "[ ]".

- %version

  Specifies file version string. You can use %s within the brackets "[ ]".

- %comments

  Specifies any comments to be associated with a particular revision of a file. You can use %s within the brackets "[ ]".

- %overwrite

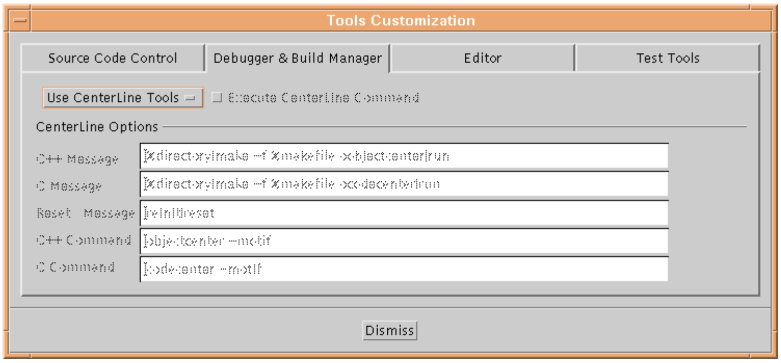
  Boolean flag used for checking out files. If True, the contents of the brackets "[ ]" are substituted on the command line. If False, nothing is added.

- %keep

  Boolean flag used when checking in files. If True, the contents of the brackets "[ ]" are substituted on the command line. If False, nothing is added.

Debugger & Build Manager

Displays the Debugger & Build Manager dialog:



C++ Message: %directory|make -f %makefile ;objectcenter|run
C Message: %directory|make -f %makefile ;codecenter|run
Reset Message: reinit|reset
C++ Command: objectcenter -motif
C Command: codecenter -motif

Figure 54. Debugger & Build Manager Panel of the Tools Customization Dialog

---

Note: The Debugger & Builder dialog is available only when you configure Builder Xcessory to use a development environment tool.

Scroll menu options

The Debugger & Build Manager menu allows you to select your debugger, as follows:

- Use Environment Tools

  Use your environment's debugger. If you start Builder Xcessory from within your development environment, the debugger option allows you to choose this option.

- Use CenterLine Tools

  Use CenterLine's debugger. Choose this option if you use either CodeCenter or ObjectCenter for application development and debugging.

  Note: The selected debugger becomes the default in your next Builder Xcessory session. For example, if you start Builder Xcessory from within your environment and select Use Environment Tools, the default debugger option is Use Environment Tools when you start your next Builder Xcessory session from within your environment. If you start Builder Xcessory outside your environment (on the command line), the Use Environment option becomes the Unset option.

- Unset

  Unsets the debugger option.

- Execute CenterLine Command toggle

  Instructs Builder Xcessory to try to start the selected CenterLine debugger. This toggle button is insensitive unless you choose Use CenterLine Tools. If unset, Builder Xcessory assumes that the Centerline debugger is already running and tries to connect.

CenterLine Options

You can set the following Centerline options:

- C++ Message

  Commands to send to CenterLine to start a debug session on C++ code.

- C Message

  Commands to send to CenterLine to start a debug session on C code.

- Reset Message

  Commands sent when Builder Xcessory switches from Debug Mode to Build or Play Mode.

- C++ Command

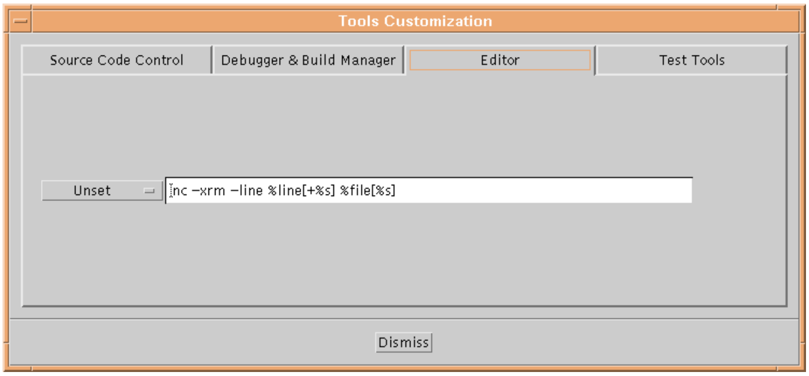
  Command used to start CenterLine debugger if necessary. The default is `objectcenter-motif`. This command is used with C++ code.

- C Command

  Command used to start CenterLine debugger if necessary. The default is xcodecenter. This command is used with C code.

Editor

Click on the Editor tab to display the Editor panel of the Tools Customization dialog:



Unset `nc -xrm -line %line[+%s] %file[%s]`

Figure 55. Editor Panel of the Tools Customization Dialog

The following table describes the available editors and their respective command line options:

Editor	Command Line Options
Unset	Indicates no editor is selected.
Nedit	nc -xrm -line %line{+%s] %file[%s]
Environment editor	Uses your development environment's editor.
Emacs	emacsclient %line[+%s] %file[%s]
VI	xterm -e vi %line[+%s] %file[%s]
User Specified	Enter the command of your choice according to the option specifications described in the following section, *"Options syntax in Builder Xcessory"* on page 106.

Using emacsclient

Builder Xcessory uses the tool `emacsclient` to tell Emacs to load a file and display the buffer. Emacs must be running. Builder Xcessory issues a request of this "Edit Server". By default, Emacs is not configured to run as a server. In order for the `emacsclient` tool to work, you must add the following Emacs Lisp command to your `.emacs` file (the file that `emacs` runs at startup):

```
(server-start)
```

Enter this line exactly as shown, including parentheses. Once Emacs loads and executes the `.emacs` file again (usually when it starts), it recognizes edit requests from `emacsclient`.

Running Emacs with `emacsclient` requires that the `emacsclient` executable and Emacs executable be run on the same machine. The socket that `emacsclient` uses to communicate with `emacs` is a local socket.

Using gnuclient

To enable `emacs` to run on one system and the client program to make edit requests from another system, you can use `gnuclient. gnuclient` works similarly to `emacsclient`, but also includes the following features:

- The `-q` option forces `gnuclient` to send its edit request but not wait for the request to complete. The `gnuclient` process exits as soon as it sends the edit request.
- The `gnuclient` software includes the ability to specify another host on which `emacs` is running and to make the edit request on that machine.

The `gnuclient` software is available with FTP from most GNU Emacs Lisp archive sites. To use `gnuclient/gnuserv`, include the following in the `.emacs` file:

```
(load "gnuserv")
(server-start)
```

Options syntax in Builder Xcessory

Builder Xcessory executes the text editor commands by constructing a command as it would be entered at a shell prompt. The command lines for the various commands use symbolic representations for the various user-specified command line options.

Builder Xcessory substitutes the appropriate value for the representation when it builds the command line. In general, the options syntax is as follows:

```
%option_name[actual_option_flag]
```

Everything within the brackets "[ ]" is substituted for the option name in the command line. Most options may also include "%s" within the brackets "[ ]", indicating where to insert dynamically determined text, (such as a file name or a line number).

Example

For example, assume a command to edit a file was "editor" and it takes the name of the file to edit as an argument. Enter the command as follows:

```
editor %file[%s]
```

If the filename argument must follow the flag "-f", enter the command as follows:

```
editor %file[-f %s]
```

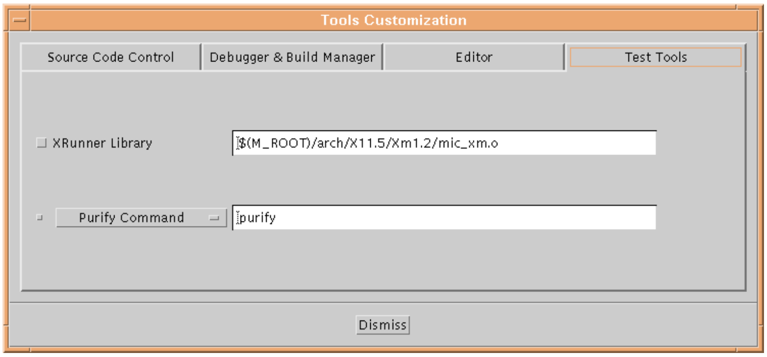
Command line option substitutions

The following list describes the command line option substitutions available for text editor commands:

- `%line`
  Specifies the line of the specified file to go to when the file is opened. %s indicates the line number to which to move the cursor.
- `%file`
  Specifies the name of the file to open. %s indicates the position of the filename.

Test Tools

Click on the Test Tools tab of the Tools Customization dialog to display the Test Tools panel:



XRunner Library: $(M_ROOT)/arch/X11.5/Xm1.2/mic_xm.o

Purify Command: purify

Figure 56. Test Tools Panel of the Tools Customization Dialog

XRunner Library

Setting the XRunner Library toggle specifies the library that your application is linked to for XRunner to communicate with the application.

When you compile, append xrunner to the standard make command, as follows:

```
make -f <makefile> xrunner
```

When using Imakefiles generated by Builder Xcessory, enter `make xrunner` (uses the last generated Makefile):

When the toggle is set, XRunner targets are added to `makefile`, so that the XRunner libraries are built into the executable. When you run XRunner outside of Builder Xcessory, XRunner will query your application according to this setting.

Note: XRunner integration is supported for versions of XRunner up to and including 2.0.

Purify Command

Setting Purify Command specifies the command which Builder Xcessory places in front of the compile line in the generated C or C++ Makefile for the application. When you compile, append pure to the standard make command, as follows:

```
make -f <makefile> pure
```

For the correct Imakefile enter `make pure` (uses the most recent Makefile generated from the Imakefile):

When the toggle is set, the “pure” target is added to `makefile`, so that the Purify commands are built into the executable. The application validates memory use and reports when a memory violation occurs. Purify will also report any leaked memory.

MemAdvisor

Setting MemAdvisor Command specifies the command which Builder Xcessory places in front of the compile line in the generated C or C++ Makefile for the application. When you compile, append pure to the standard make command, as follows:

```
make -f <makefile> pure
```

For the correct Imakefile enter `make pure` (uses the most recent Makefile generated by Imakefile):

When the toggle is set, MemAdvisor targets are added to `makefile`, so that the MemAdvisor commands are built into the executable. The application validates memory use and reports when a memory violation occurs. MemAdvisor will also report any leaked memory.

GIL Import Preferences

Displays the GIL Customization dialog.

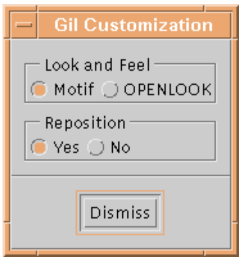


Figure 57. GIL Customization Dialog of the Options Menu

Interface files created with Devguide may be imported into Builder Xcessory, allowing you to incorporate existing GIL (Guide Interface Language) interfaces into interfaces you create with Builder Xcessory.

The GIL Customization dialog allows you to set the following GIL to UIL converter options:

Look and Feel Set the converter to produce either Motif or OpenLook appearance and behavior. Set to Motif by default.

Reposition When set to Yes, the converter will attempt to reposition objects where appropriate. When set to No, reposition is not attempted. Set to Yes by default.

Managers Menu

Managers help you keep track of the many procedures, types, identifiers, constants, and resource styles that you define in a large, complex interface. The Browser Manger menu allows you to view Builder Xcessory managers. Select Managers from the Browser menu bar, to display the following menu:

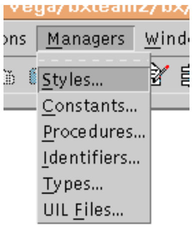


Figure 58. Browser Managers Menu

The managers accessible from this menu allow you create, edit, and delete elements of callback and creation routines. Changes entered in the Callbacks Editor or the Creation Routine dialog are reflected in the managers, and vice-versa.

Refer to *"Managers"* on page 161 for detailed information about each of the Builder Xcessory managers.

Windows Menu

Allows you to deiconify and raise Builder Xcessory main windows, top level shells, and dialog shells in your interface.

Select Windows from the Browser menu bar to display the Browser Windows menu:

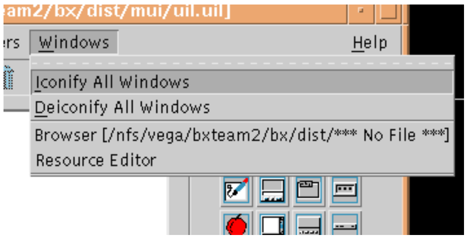


Figure 59. Windows Menu

The Windows menu contains entries for each of Builder Xcessory's main windows, and any top level shells and dialog shells in your interface.

- Select Iconify All Windows or DeIconify All Windows to iconify or deiconify all Builder Xcessory windows and all of your application's windows.
- Select Browser, or Resource Editor from the Windows menu to raise the item to the top of your interface

Pallette

Pallette Options

Depending on the currently selected language and the platform on which you are running Builder Xcessory, the Palette icons correspond to the following objects:

Palette object icons

- Motif Xm widgets
- ViewKit ObjectPak Vk objects
- EnhancementPak Xi widgets
- DEC DXm widgets (available only on DEC UNIX platforms)
- SGI Sgm widgets (available only on SGI platforms)
- Common Desktop Environment (CDE) Dt widgets (available only on platforms with CDE and BX CDE support)

Palette object icons table

The following table lists the objects displayed for specific platforms and languages:

Objects	Platforms	Languages
Motif Xm widgets	All	C++, C, ViewKit, and UIL
EnhancementPak Xi widgets	All	C++, C, ViewKit, and UIL
ViewKit ObjectPak Vk objects	All	ViewKit only
SGI Sgm widgets	SGI	C++, C, ViewKit, and UIL
CDE Dt widgets	AIX, Solaris, HP-UX 10+, and DEC UNIX 4	C++, C, ViewKit, and UIL

Other Palette collections

The Palette can also include user-created collections of objects, other widget sets, and user-created classes. Refer to *B Palette Objects* and for more detailed information about the available objects. The following sections show the Palette groups displayed according to the language you select.

Motif Widgets

The Motif widgets are displayed on the Palette for all platforms and for all languages:

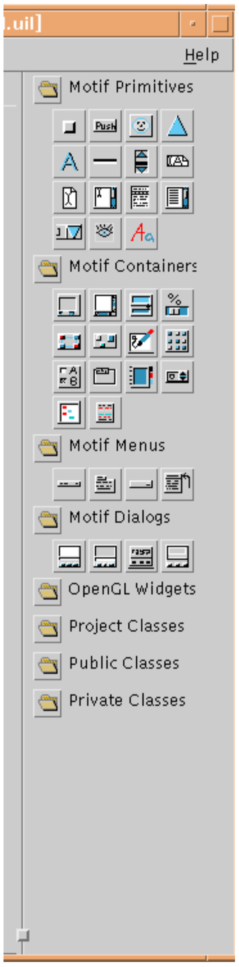


Figure 60. Motif Palette Icons (for C, C++, ViewKit or UIL)

EnhancementPak Widgets

BX PRO users

By default, the BX PRO Palette includes the EnhancementPak widgets.

Builder Xcessory users

Start with EPak Widgets must be set on the Behavior tab of the User Preferences dialog to display the EnhancementPak icons. See *"Behavior toggle options"* on page 93 for more detailed information.

Note: You can use the EnhancementPak widgets in your interface, but you must purchase the EnhancementPak library to compile any interface built with the EnhancementPak widgets. Contact your ICS Sales Representative for more information.

The EnhancementPak widgets are displayed on the Palette for all platforms and for all languages:

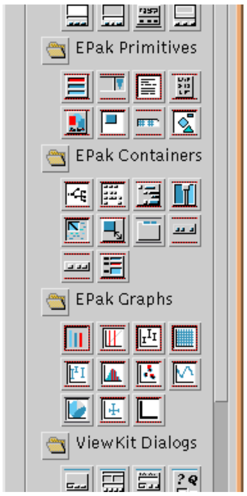


Figure 61. EnhancementPak Palette Icons (for C, C++, ViewKit, or UIL)

ViewKit ObjectPak Classes

Note: If you are using BX PRO, you can use and compile the ViewKit ObjectPak objects in your interface. If you are using Builder Xcessory, you can use the ViewKit objects, but you must purchase the ViewKit ObjectPak library to compile any interface built with ViewKit ObjectPak objects. Contact your ICS Sales Representative for more information.

The ViewKit ObjectPak classes are added to the Palette for all platforms, but only when you select ViewKit as the current language:

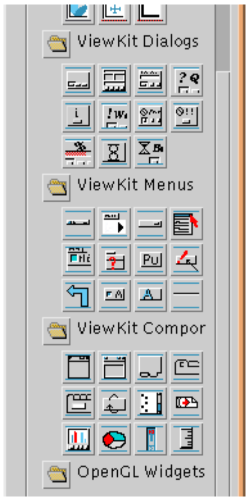


Figure 62. ViewKit Palette Icons (for ViewKit only)

Palette Groups

The Palette is divided into several groups that consist of objects grouped by related functionality.

Manipulating groups and icons

You can add, delete, and rearrange groups, and move Palette collections between groups. You can drag an icon from one group to another, or you can press MB3 over an icon and Edit the icon properties group, name, and pixmap.

Nesting groups

You can nest groups by creating a new group as part of another group. For example, create a “ViewKit’ group and move all ViewKit-related groups into the “ViewKit” group. You can then hide all the ViewKit items by closing just the single group.

Hiding/ displaying groups

To hide a group, click on the folder icon to the left of the group name. Click again on the icon to restore the view of the group.

Creating subordinate groups

To create a new group subordinate to an existing group, select New Group from the Palette MB3 Quick Access menu.

Groups common to all Palettes

In addition to the platform- and language-dependent groups, the Palette displays the following groups:

- Project Classes

  High-level UI objects you create or for a project when you open a project save file.

- Private Classes

  High-level UI objects that are stored in the following directory:

  `${HOME}/.builderXcessory6/classes` directory

  These objects appear each time you use Builder Xcessory.

- Public Classes

  High-level objects that are stored in the following directory:

  `{BX}/xcessory/classes` directory.

  These objects appear on every Builder Xcessory user’s Palette.

Motif Groups

For C, C++, and UIL, the Motif and platform-specific Palette objects are grouped initially as follows:

- Primitives

  XmToggleButton, XmPushButton, XmDrawnButton, XmArrowButton, XmLabel, XmSeparator, XmScrollBar, XmTextField, XmText, XmScrolled-Text, XmList, XmScrolledList, XmComboBox, and XmIconGadget.

- Containers

  XmMainWindow, XmScrolledWindow, XmPanedWindow, XmScale, XmForm, XmBulletinBoard, XmDrawingArea, XmRowColumn, XmRa-dioBox, XmNotebook, XmSpinBox, XmContainer, and XmFrame.

- Menus

  XmMenuBar, XmPulldownMenu, XmOptionMenu, and XmPopupMenu.

- Dialogs

  XmSelectionBox, XmFileSelector, XmMessageBox, and XmCommand.

See *"Motif Xm Widgets"* on page 283 for more detailed information about the Motif widgets.

ViewKit ObjectPak Groups

The ViewKit ObjectPak Classes are added to the Palette when you select ViewKit as the current language. Initially, ViewKit classes are grouped in the following folders:

- Dialogs

  VkGenericDialog, VkFileSelectionDialog, VkPromptDialog, VkQuestionDialog, VkInfoDialog, VkWarningDialog, VkErrorDialog, VkFatalErrorDialog, VkPro-gressDialog, VkBusyDialog, and VkInterruptDialog.

- Menus

  Menu Bar, Sub Menu, Option Menu, Popup Menu, Radio Sub Menu, Help Pane, Menu Action, Menu Confirm First Action, Menu Undo Manager, menu Toggle, Menu Label, and Menu Separator.

- Components

  VkSimpleWindow, VkWindow, VkTabbedDeck, VkTabPanel, VkRepeatBut-ton, VkOutline, VkCompletionField, VkGraph, VkPie, VkVuMeter ,VkTab-Set, and VkTickMarks.

See *“ViewKit ObjectPak Vk Classes”* on page 300 for more detailed information about the ViewKit classes.

EnhancementPak Groups

Note: Start with EPak Widgets must be set on the Behavior tab of the User Preferences dialog to display the EnhancementPak icons. See *“Behavior toggle options”* on page 93 for more detailed information.

For all languages, the following EnhancementPak widget groups are added to the Palette:

- EPak Primitives

  XiColorSelector, XiCombinationBox, XiExtended18List, XiFontSelector, XiIconButton, XiPanner, XiDataField, and XiPixmapEditor.

- EPak Containers

  XiTree, XiIconBox, XiOutline, XiPaned, XiPorthole, XiStretch, XiTabStack, XiToolbar, XiButtonBox, and XiColumn.

- EPak Graphs

  XiBarPlot, XiCallbackPlot, XiErrorPlot, XiFadePlot, XiHighLowPlot, XiHistoPlot, XiImagePlot, XiLinePlot, XiPiePlot, XiTextPlot, and XiPlotter.

See *“EnhancementPak Xi Widgets”* on page 314 for more detailed information about the EnhancementPak widgets.

Reference Count

Styles, constants, identifiers, procedures, and types are associated with reference counts on their respective editors. The number of references contained in the Palette collections is reported (Palette = N) on the editor. Palette references are included in the overall reference count.

Maintaining reference counts when deleting collections

Cutting a collection does not affect the reference count. To decrement the reference counts of styles, constants, and so forth contained in a collection, you must delete the collection from the Palette.

Adding and Moving Palette Collections

You can drop a widget, class, or Browser instance name onto a Palette group to add the widget and all of its descendants to that group as a new collection.

Using drag and drop to add and move collections

Use the following methods to add or move collections:

- Dropping onto a Palette group’s name adds the new collection to a closed group.
- Dragging a collection from one Palette group and dropping it onto another group moves the collection to the end of the group. Dropping the collection on an existing collection inserts the dragged collection in the group immediately before the collection onto which it was dropped.

Moving collections to offscreen groups

If you want to move a collection to a group which is currently offscreen, use one of the following methods:

- Display the Palette as Labels Only.
- Hide the collections of the other Palette groups.
- Edit the collection and enter a new group in its Group field.
- Use the Cut/Paste items from the Palette MB3 Quick Access Menu.

Note: You can also move a Palette collection to a blank space within the same group.

Hiding and Displaying Palette Groups

To hide and display Palette groups, use the following methods:

Displaying

- Click MB1 on a group’s closed folder to expand the Palette to display the collections in that group.

Hiding

- Click on a group’s open folder to hide the collections of that group.

MB3 Quick Access Menu

To invoke the MB3 Quick Access Menu, press MB3 while the mouse pointer is in the Palette window.

Note: The MB3 convenience menu is active only when the pointer is positioned over a Palette icon or a Palette group name.

The MB3 menu enhances functions of the Catalog and Edit menu as follows:

Cut Allows you to place the selected Palette object or group into a paste buffer.

Paste Allows you to paste the object or group currently in the paste buffer. Pasting a group will nest it within the group where the cursor is positioned. Pasting an item will insert the item immediately before the item under the cursor or at the end of the group if the cursor is over the Group Label.

Delete Deletes the selected Palette object or Palette group.

New Group Displays the System Catalog dialog:

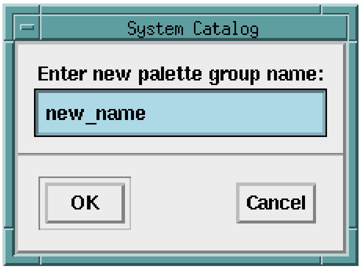
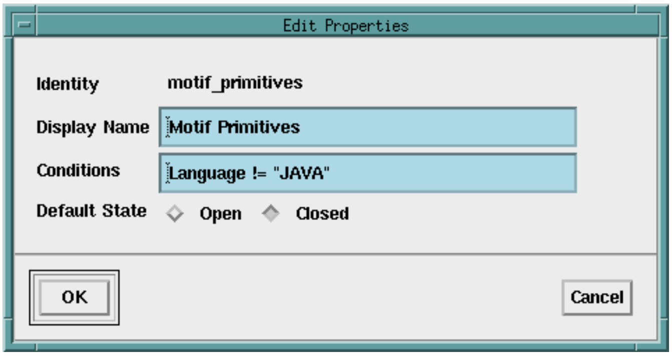


Figure 63. System Catalog Dialog

To create a new Palette group, enter the new name in the text field.

Properties Displays the Edit Properties dialog box. Depending on the object or group you select, displays several editable text fields. In the following example, the Motif Primitives group is selected:



Identity motif_primitives
Display Name Motif Primitives
Conditions Language != "JAVA"
Default State Open [✓ Closed]

Figure 64. Edit Properties Dialog Box with Motif Primitives Group Selected

Identity Shows the name that identifies the Palette group or Palette object, in this example motif_primitive (this field is not editable).

Display Name Shows the name of the Palette object or Palette group as displayed on the Palette, in this example, Motif Primitive. Click MB1 inside the Display Name text field and enter a new name for the Palette group.

Conditions Shows the conditions required for the Palette Group. Both the item and the group can optionally name conditions under which they should appear on the Palette.

The valid conditions for the Items name attributes of the system on which Builder Xcessory is running and what other software it is using. You can specify a value SystemAttribute(tag), where "tag" is one of the following values:

Tag	**Explanation**
DXm	Digital enhanced OSF/Motif
EPak	ICS EnhancementPak widget set
IrisGL	SGI IrisGL library
OpenGL	SGI OpenGL library

You can join multiple uses of SystemAttribute with "&&" (logical AND) and "||" (logical OR) and group them in parentheses.

The valid conditions for the Groups tag name similar attributes of the system on which Builder Xcessory is running, with additional possibilities of the form "tag operator string-value", where tag is one of the following values:

Tag	**Explanation**
DatabaseName	Test value of resource databaseName
Env(var)	Test value of environment variable "var"
Language	Test language BX is using
Platform	Test operating system BX is running on, using function uname(2V)
Version	Test operating system version (release), using function uname(2V)

Operators

You can use the following operators;

- ==
- !=
- <
- >
- >=
- <=

The operators that test greater-than (>) or less-than (<) are appropriate only for tests of Version. You can join multiple uses of these expressions, along with uses of SystemAttribute, with the logical operators in this list. The catalog file should be in the `{BX}/xcessory/package/` directory.

Default State

Toggle buttons that define whether the Palette Group is displayed when initially loaded into a Palette.

Adding User-defined Widgets and Objects

You can add your own Xt Intrinsics-based Motif 2.1.x/, C++ classes, or ViewKit components to Builder Xcessory. User-defined widgets and objects appear on the Palette, and you can access and manipulate user-defined widgets and objects as you would any other Palette collection.

Refer to *Chapter 2—Adding Widgets* in *Customizing Builder Xcessory* for more detailed information about adding widgets to the Palette.

Resource Editor

3

Overview

The Resource Editor allows you to view and modify resources for classes, class instances, and widget instances.This chapter describes the following components of the Resource Editor, and provides an overview of extended editors and expressions:

Updating the Resource Editor

To update the Resource Editor, use one of the following methods:

- Double-click on the object in the Browser.
- Click on the Update icon in the upper right corner (just below the menu bar) of the Resource Editor.
- Select Update from the Resource Editor View menu.

The Resource Editor displays different information depending on whether the update is for a widget instance, class instance or a class, as described in the following sections.

Resource Editor for a Widget Instance

When you select a widget instance, the Resource Editor displays the text fields and resources described in the following sections.

Resource Editor Text Fields for a Widget Instance

When updated for a widget instance in Instances View, the Resource Editor displays the following text fields:

Instance Name	Displays the widget instance name, as it appears on the Browser.
Class Name	Displays the Widget Class to which the widget instance belongs.
Style	Displays the name of the resource style applied to the widget instance. If no resource style has been applied to the widget, this field is empty. See *“Style”* on page 140 and refer to *“Style Editor”* on page 174 for more detailed information about style resources.

Resource Settings for a Widget Instance

The option menu to the far right of each resource text field enables you to change the resource setting for each resource. The following table lists and describes the available resource settings:

Resources	**Description**
None	The resource for this widget is set to the default value of that widget class. When a widget instance is first created, all of its resources have a resource setting of None.
Code	The value is hardcoded, and the user cannot change the value.
App **Note:** This setting is not available for Java.	The resource has been set as an app-default. When the app-defaults file is written, it includes all resources whose app-defaults value is set as App. The user can change an app-default resource value by editing the app-defaults resource file or by overriding it from a local resource file or from the command line (with the toolkit option “-xrm”). Typically, the app-defaults file is shipped with your finished application so that the application user can run your application without having specified anything on the command line.
Const	The value is set to a constant, selected by clicking the arrow button to the right of Const and selecting a constant from the pull-down menu. The menu contains all constants defined for the interface. Selecting “New” displays the Constant Editor with a new constant. **Note:** “New” is not available for pixmap constants.
Ident	The value is set to an identifier, selected by clicking the arrow button to the right of Ident and selecting an identifier from the pull-down menu. The menu contains all identifiers defined for the interface. Selecting “New” displays the Identifier Editor with a new constant.
Expr	The resource takes an expression as its value. The value is hard-coded and cannot be overridden by the application defaults. See *“Expressions”* on page 160.
Style	The resource is one of a set of resources in the style that has been applied to the widget.
Expose **Note:** For some resources in Java and in ViewKit, the expose option is insensitive.	You can view and modify a widget instance’s resource when the Resource Editor is updated for the class instance in Instances View. Use this resource in conjunction with any other setting except None and Style. **Note:** The Expose setting is available only to widget instances which are members of a class.

Widget Instance Resources

If the selected widget is a Motif or ViewKit object, the Resource Editor displays the Widget Instance resources and callbacks.

Constraint resources for Motif and ViewKit objects

Children of some container widgets add constraint resources at the end of the resources list (the word “Constraint” precedes these resource names). For example, if you place a Button in a Form container, additional resources are displayed which allow you to control the layout of the Button within the Form. However, if you place a Button in a BulletinBoard container, no resources are added to the Resource Editor.

Constraint resources for Java objects

If the selected widget is a Java object, the Resource Editor displays a list of attributes and events associated with the selected object. Relevant layout attributes are also displayed.

For example, when you set the object’s layout to GridBagLayout, the Resource Editor also displays resources associated with the GridBag constraint object. The GridBag constraint object is part of the code generated by Builder Xcessory, but is not displayed on the Palette. The layout resource appears with the container if the attribute controls the overall layout and with the child if the attribute controls the placement of a particular child.

Code generated Java objects

In some cases, non-visual objects are required to set resources. The following objects are not on the Palette, but are created when Builder Xcessory generates code:

- BorderLayout
- FlowLayout
- CardLayout
- GridLayout
- GridBagLayout
- GridBagConstraints
- CheckBoxGroup

These objects supply additional resources in the Resource Editor for the associated object.

Resource Editor for Class Instances and Classes

The following sections describe the text fields displayed in the Resource Editor when you select a class instance or a class.

Resource Editor Text Fields for Class Instances

When updated for a class instance, the Resource Editor displays the following text fields:

Note: These text fields are automatically updated for the currently selected object, regardless of whether resource information has been updated.

Instance Name Displays the class instance name, by default the UI Class Name, with the first letter of the name in lowercase.

Base Class Displays the base name to which the C++ Source and Header suffixes will be appended. The File Base Name is set to the Class Name by default.

Derived Class Displays the Builder Xcessory class name. By default, the Instance Name appended with Derived.

Note: Builder Xcessory generates and displays derived files only if you set Generate Derived Files on the Code Generation tab of the C++ Language Settings dialog. Refer to the section*“Toggle options”* on page 64 for more information.

Note: Derived classes are provided for backward compatible code generation only. We do not recommend using Derived Classes unless you want backwards compatible code generation to Builder Xcessory versions previous to Builder Xcessory version 3.5.

Class Widget Resources

Below the text fields, the Resource Editor displays the Class Widget Resources. That is, any resources in a constituent widget of the class which have been set as Expose. These resources are identified as follows:

```
<widget instance name>.<resource name>
```

Example

For example, assume Class1 is composed of a main window with one child, a button box that has four push button children. In Classes View you set the backgrounds of the first two push buttons to red and blue, respectively, and set the button box's fill option to XiFill None. The resource setting of each resource is Code, but you also select Expose so that the values will be modifiable within a given instance of the class.

When you update the Resource Editor for class1, an instance of Class1, the following class widget resources are displayed:

Resource	Value	Setting
buttonBox.fillOption	XiFillNone	None
pushButton.background	red	None
pushButton1.background	blue	None

Note: Java class instances are the same as Motif class instances. However, the <resource-name> is replaced with the name of a Java class attribute.

Resource Editor Text Fields for Classes

In Classes View, when you select a class, the Resource Editor displays the following text fields:

Note: These text fields are always automatically updated for the currently selected widget, regardless of whether resource information has been updated.

Base Class

Name of currently selected Builder Xcessory class. Builder Xcessory enforces convention of single name with a capital letter.

Derived Class

Name of currently selected Builder Xcessory derived class. Builder Xcessory enforces convention of single name with a capital letter.

Builder Xcessory generates and displays derived files only if you set Generate Derived Files on the Code Generation tab of the C++ Language Settings dialog (from the Browser Options menu).

Note: Derived files are supported for C++ code generation option only as a backward compatibility option, and should not be used for new development.

Derived Class File

Base name to which the derived class C++ source and header files will be appended. The default value is the Derived Class. For example, files generated from Derived Class File “foo” will have the default names FooDerived.C (for the source file) and FooDerived.h (for the header file). By default, not displayed.

Note: Builder Xcessory generates and displays derived files only if you set Generate Derived Files on the Code Generation tab of the C++ Language Settings dialog (from the Browser Options menu).

Resource Editor for Multiple Widgets or Class Instances

When you select multiple widgets, the Resource Editor displays the same text fields and resources as described in Instances View. The following sections describe the differences between views.

Note: These text fields are always automatically updated for the currently selected widget, regardless of whether resource information has been updated.

Resource Editor Text Fields for Multiple Widgets or Class Instances

The following sections describe the text fields displayed in the Resource Editor for multiply-selected object instances.

Instance Name

Displays a “!=” button to the left of the Instance Name text input field. Because multiple object instances are selected, each instance has a different name and all of the names cannot be displayed in this field. Click on the “!=” button to display the Multiple Header Editor, which lists the name of every object instance that is currently selected in the Instance column.

Class Name

The Class to which the object instances belong, if all of the selected instances belong to the same Class. Otherwise, a “!=” button is displayed to the left of the Class Name text input field.

If each instance belongs to a different Class, all of the Classes cannot be displayed in this field. Click on the “!=” button to display the Multiple Header Editor for a list of widget instance names currently selected in the Instance column, and each associated Widget Class in the Value column.

Note: Selecting one value from this list and then clicking on Apply sets the Class Name of all selected objects to that value.

Style

Displays the name of the resource style applied to the object instances, according to the following conditions:

- If a resource style has not been applied, this field is empty.
- If the selected objects have different styles, a “!=” button is displayed to the left of the Style text input field.
- If each instance uses a different Style, all the Styles cannot be displayed in this field. Click on the “!=” button to display the Multiple Header Editor for a list of widget instance names currently selected in the Instance column, and each associated resource style in the Value column. See *“Styles”* on page 166 for more detailed information about resource styles.

The value specified in Style is set in the appropriate resource in the objects which compose the class instances in your interface. Such a value might be overridden in the class instance, depending on its resource setting.

Resource Settings

The option menu to the far right of each resource text field enables you to change the resource setting for each resource. The following table list, and describes the available resource settings:

Resource	**Description**
None	Resets the resource value to the default.
Code	The resource value is hard-coded and cannot be overridden by the application defaults or the X Resource Database.
App **Note:** This Resource Setting is not applicable for Java.	The resource value is written to a defaults file and can be overridden by the X Resource Database.
Const	The resource takes a constant as its value. Selecting this setting displays a list in which every existing constant of the appropriate type is displayed. The value is hard-coded and cannot be overridden by the application defaults. Selecting “New” displays the Constant Editor with a new constant **Note:** “New” is not available for pixmap constants.
Ident	The resource takes an identifier, chosen from a displayed combination box, as its value. The value is hard-coded and cannot be overridden by the application defaults. Selecting “New” displays the Identifier Editor with a new constant.
Expr	The resource takes an expression as its value. The value is hard-coded and cannot be overridden by the application defaults. See *“Expressions”* on page 160 for more information.
Style	The resource is one of a set of resources in the style that has been applied to the widget. Unless the resource has been set with a style, this option is not present on the option menu.
Expose **Note:** For some resources in Java and ViewKit, the expose option is insensitive	You can view and modify a widget instance’s resource when the Resource Editor is updated for the class instance in Instances View. This resource can be used in conjunction with any other setting except None and Style.

When you set a resource value either directly or through an extended editor (see *“Extended Editors”* on page 159), the app-defaults value is set to the appropriate resource type as set on the Default Resource Placement dialog. For further details, refer to the section *“Default Resource Placement (Ctrl + M)”* on page 154.

Class Member Editor

In Classes View, the Resource Editor displays the Class Members of the currently selected class instead of resources. Class Members are distinguished by their type (Data or Method) and scope (Public, Private, or Protected). Use the Class Member Editor to specify multiple members in each category, and add or edit Members.

Click the (...) button to the right of the Class Methods and Data text field to display the Class Member Editor:

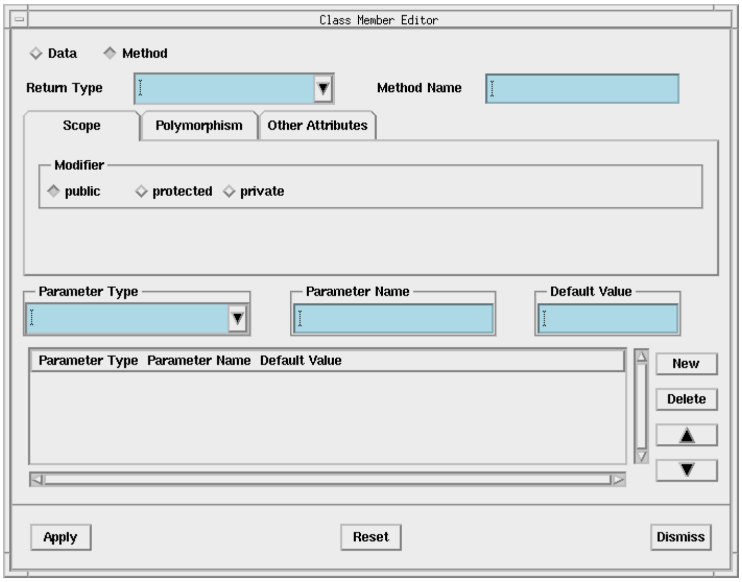


Figure 69. Class Member Editor for a Class Method

Data and Method toggles

Use these toggles at the top of the Class Member Editor to select Data or Method mode.

Other fields and toggles allow you to assign the type, name, and other attributes such as the member’s scope.

Note: The attribute selections available on the Scope, Polymorphism, and Other Attributes tab panels vary depending on the currently selected code generation language.

Some cross-checking of selections occurs so that inappropriate selections are disabled. For example, if you are using C++ code generation, an initial static value is unavailable for methods and is disabled.

Parameter fields

Parameter fields are available only when you select Method mode. For Data mode, the parameter fields are hidden. Static data members can provide initial values. For methods, the parameter fields allow you to enter argument types, parameter names, and, for C++ code generation, default values.

Note: The Class Member Editor is a valuable tool for specifying language-specific attributes to each added class member. However, although some cross-checking is performed to confirm the compatibility of your selections, the cross-checking is not exhaustive and is not intended to be a robust language checker.

User Code Blocks, methods, and data members

Member methods and data can also be inserted directly into the User Code Blocks of the class file. These edits are not visible or editable in the Resource Editor. Because they are inside User Code Blocks, they are retained when the files are regenerated.

Component Menu

Allows you to hide or show an object, make a widget into a gadget, rename the topLevelShell’s creation routine, and re-specify the creation routine parameters.

Select Component from the Resource Editor menu bar to display the Component menu:

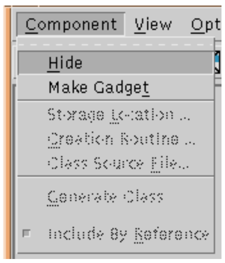


Figure 70. Component Menu

Hide/Show

The object is displayed during your Builder Xcessory session. By default, Show is set. When you select Hide, the object is immediately removed from your display. A hidden object will not be shown in the generated code.

Make Gadget/Widget

Allows you to toggle between the widget and gadget variants of various Motif Primitives. The Class Name, as it appears on the Resource Editor, is updated dynamically.

Note: Make Gadget is insensitive if Java is the selected language.

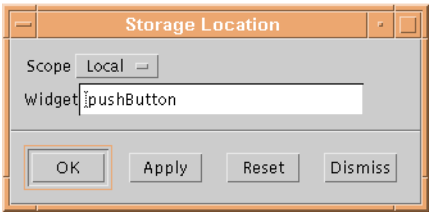
Storage Location

Note: This feature is applicable for C only. In C++, object instances will be protected data members of a C++ class.

C U

Builder Xcessory allows you to declare widgets outside the scope of the creation file. These widgets are accessible from anywhere within your program. While this is not recommended programming practice, we recognize that there are instances in which this capability is useful.

Select Storage Location from the Component menu of the Resource Editor to display the Storage Location dialog:



Scope Local
Widget pushButton

Figure 71. Storage Location Dialog from the Component Menu

Scope

Set the Scope text field to one of the following:

- Local

  Default Widget ID Scope type. The widget is declared within the scope of the creation routine and is accessible only within the scope of the creation routine.

- Global

  Creates a globally defined widget ID for simple instances or a widget structure for class instances. The declaration is placed outside of the creation routine in a definition file, allowing you to access the widget ID from anywhere else within the program.

  Definitions are written to the definition file, `creation-c.h` for C, or `main-uil.h` for UIL. You can then include these files in other modules requiring access to these widget IDs.

- Other

  Assumes that your entry in the Widget ID Scope text field (with the widget instance name again the default) is defined in outside of the creation file. Also assumes that the value is accessible from within the creation routine.

  For example, setting Widget ID Scope type to Other and specifying `param->one` in the Widget ID Scope text field outputs a declaration of the form:

  ```
     param->one = XtCreateWidget(...)
  ```

  where `param` is assumed to be a pointer to a structure with member one of type Widget. Using this method, you can pass a pointer to a structure of all your widget IDs to the creation routine and make direct assignments to the structure members.

Widget

Specify widget ID names in the Widget text field for widget instances. The widget instance name is used as the default.

Class instances

For class instances, the Widget label changes to the name of a structure that contains all widget IDs in that class instance. You can specify any ID name in the Widget text field.The class instance name is used as the default.

Note: When editing classes in Classes View, Storage Location is insensitive. You are editing a template for a hierarchy of widgets rather than the widgets themselves. You must set Storage Location on a class instance in Instances View.

Storage location buttons

The following table lists and describes the buttons on the Storage Location dialog:

Button	**Description**
OK	Sets the specified storage location and widget ID name for the currently selected widget, and dismisses the dialog.
Apply	Sets the specified storage location and widget ID name for the currently selected widget.
Reset	Reads the last widget ID and storage location for the currently selected widget into the dialog.
Dismiss	Removes the Storage Location dialog.

C++, ViewKit, and Java Generation

Changing Storage Location does not alter the C++, ViewKit, or Java code generated by Builder Xcessory. All widget instances are automatically declared as protected members of the class and are accessible from within class instances. If you want to access these widget instances, add a public method to the class definition.

C Generation

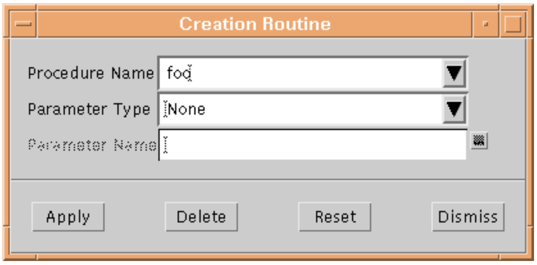
When you set Storage Location for a class instance, if you generate C code, the structure itself is passed for types Local and Global. A pointer to the structure is passed for type Other (the location must be valid).

UIL Generation

In contrast to C generation, no structure is generated for class instances when you generate UIL. Only the widget ID of the uppermost widget instance in the class instance is stored. You can access other widgets using the XtNameToWidget call. Widget instances which are not within classes are generated just as in C.

Creation Routine

Select Creation Routine on the Component menu of the Resource Editor to display the Creation Routine dialog:



Procedure Name foo
Parameter Type None
Parameter Name

Figure 72. Creation Routine Dialog from the Component Menu

Note: The Creation Routine dialog is only available for children of topLevelShells.

A widget's Creation Routine is the routine in the `creation-c.c` file, called by the main-c.c routine, which creates the widget. You can rename this routine and specify the type and name of the parameter passed to the routine.

Procedure Name Click the arrow button to the right of the Procedure Name text field to view a combination box containing all currently defined procedure names. Enter the name into the Procedure Name text field.

Parameter Type If you enter a new Procedure Name, the default parameter type is None. If you choose an existing routine, the Parameter Type field displays the type assigned to the routine in the Procedure Manager. To change this type, the following conditions must apply:

- The procedure is not already referenced elsewhere.
- The type has not been applied to the widget.

Parameter Name The Parameter assigned to the Creation Routine can be an identifier or an actual value corresponding to the Parameter Type. New identifiers entered in the Parameter text field will be created and displayed in the Identifier Manager with the appropriate Parameter Type. Constants are not valid Creation Routine parameter values.

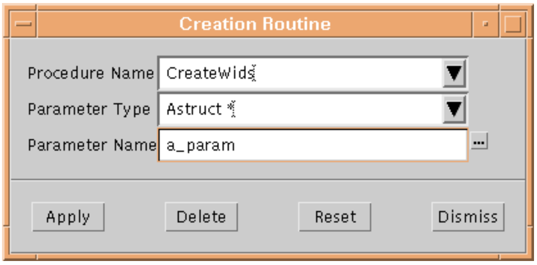
Creation Routine dialog buttons

The following buttons are on the Creation Routine dialog:

- Apply
  Sets the procedure as the creation routine.
- Delete
  Clears the contents of the Creation Routine dialog text fields.
- Reset
  Reads the last applied values back into the Creation Routine dialog.
- Dismiss
  Removes the Creation Routine dialog.

Example

For example, if you specify the following for the child of a topLevelShell:



Procedure Name: CreateWids
Parameter Type: Astruct *
Parameter Name: a_param

Figure 73. Creation Routine Example

A Creation Routine with these values generates the following function declaration:

```
Widget CreateWids(Widget parent, AStruct*
a_param)
```

You can use the Creation Routine dialog to specify parameters which pass widget IDs between callbacks.

Class Source File

Note: In Java, you cannot edit this field because the file name must match the class name.

Click on Class Source File on the Component menu of the Resource Editor to display the Class Source File dialog:

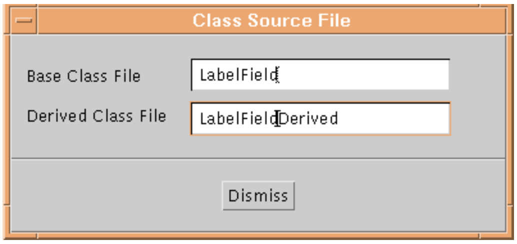


Figure 74. Class Source File Dialog

Base Class File

Base name to which the C++ and C source and header suffixes will be appended. Set to the Base Class Name by default. For example, files generated from Base Class File “Foo” will be `Foo.C` (for the source file) and `Foo.h` (for the header file).

Derived Class File

Base name to which the derived class C++ source and header files will be appended. The default value is the Derived Class. For example, files generated from Derived Class File “foo” will have the default names FooDerived.C (for the source file) and FooDerived.h (for the header file). By default, not displayed.

Note: Builder Xcessory generates and displays derived files only if you set Generate Derived Files on the Application tab of the Code Generation Preferances dialog (from the Options menu).

Generate Class

Select Generate Class to generate source code for the selected class. Set on by default. When linking a class into your application from a library, you can turn off source code generation.

Include By Reference

Select Include By Reference to forward reference a class without including its header files. Set on by default. If set off, includes the header of nested classes.

Note: This feature is available only when you select C++ or ViewKit as your current language.

View Menu

The View menu allows you to update the Resource Editor for the selected object, change the order in which resources are displayed, and display or hide the Resource Editor Toolbar, Search Bar, and Header Data text fields, as well as change the set of resources displayed on the Resource Editor.

Select View from the Resource Editor menu bar to display the View menu:

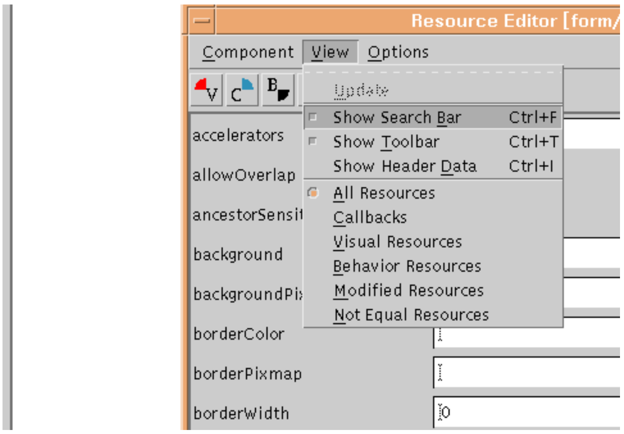


Figure 75. View Menu

The following sections describe how these toggle settings customize the set of resources displayed in the Resource Editor.

Update

Updates the Resource Editor to display the resource values for the selected object. Double-click on the instance name in the Browser or on the object in the application to update the Resource Editor.

Show Search Bar(Ctrl+F)

Adds or removes a panel at the bottom of the Resource Editor, containing the Find Resource text field and two arrow buttons.

Find Resource text field

In the Find Resource text field, specify the string you wish to search for in the Resource Editor resource list. Once you establish a search direction by clicking an arrow button or pressing enter, the list is searched dynamically as you enter text. When you type “text” the search begins by finding the first instance of “t”, then “te”, and so forth.

The search is case insensitive, wraps around the list, and is terminated with a beep and a message posted to the Browser Message Area in the event that the string is not found.

Forward search

To perform a forward search, use one of the following methods:

- Click the down arrow button
- Carriage return
- Ctrl+S

Backward search

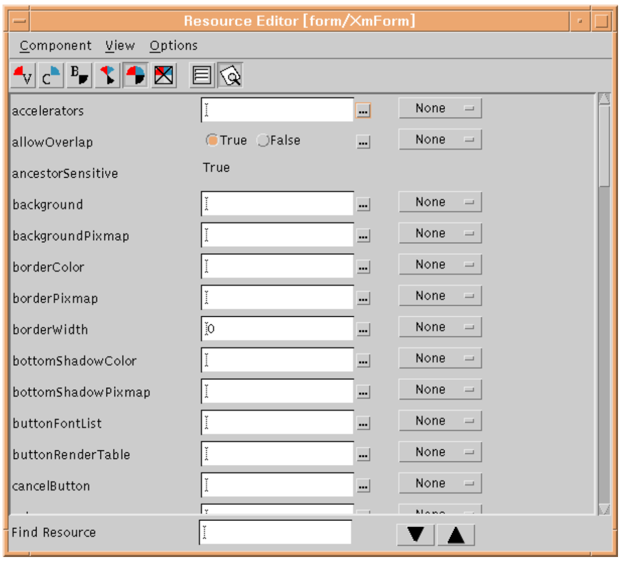
To perform a backward search, use one of the following methods:

- Click the up arrow button
- Ctrl+R

Note: Ctrl+U clears the text field.

Show Toolbar (Ctrl+T)

Displays the Resource Editor Toolbar. The Resource Editor Toolbar is displayed just below the Resource Editor menu bar.



accelerators		None
allowOverlap	[✓ True] False	None
ancestorSensitive	True	
background		None
backgroundPixmap		None
borderColor		None
borderPixmap		None
borderWidth	0	None
bottomShadowColor		None
bottomShadowPixmap		None
buttonFontList		None
buttonRenderTable		None
cancelButton		None

Figure 76. Resource Editor Toolbar

Adding a menu item to the Toolbar

Add a menu item to the Toolbar by pressing the Shift key and selecting the item from its menu. For example, to add Update to the Toolbar:

1. Confirm that the Toolbar is being displayed. If not, select Show Toolbar from the Resource Editor View menu.
2. Hold down the Shift key and select Update from the Resource Editor View menu.

Deleting a menu item from the Toolbar

Delete a menu item from the Toolbar by pressing the Shift key and selecting either the appropriate icon on the Toolbar or the item on the menu. For example, to remove Update from the Toolbar:

1. Hold down the Shift key.
2. Select Update on the Toolbar.
3. Release the Shift key.

Hide Toolbar

Hides the Resource Editor Toolbar.

Show Header Data (Ctrl+I)

Displays or hides the text fields at the top of the Resource Editor.

All Resources

The Resource Editor resource list contains all resources supported by the currently selected object.

Callbacks

The Resource Editor resource list contains all callbacks for the currently selected object.

Visual Resources

The Resource Editor resource list contains all visual resources for the currently selected object. Visual resources are the resources that determine the appearance of a widget (such as, foreground, background, and labels).

Behavior Resources

The Resource Editor resource list contains all behavior resources for the currently selected object. Behavior resources are the resources that determine how the widget acts(such as, fillOnArm, multiClick, or sensitive).

Modified Resources

When set, the Resource Editor resource list contains all resources for the currently selected object which have been modified from the default resource value for the class.

Not Equal Resources

When set, the Resource Editor resource list contains all “not-equal resources” for the currently selected objects. If you have selected two or more instances, the Resource Editor will display only the resources for which the multiply-selected instances have different values.

Options Menu

The Options menu allows you to set the Resource Editor to automatically update when a new object is selected, as well as configure the order in which the resource lists are presented and specify how to generate code for all resource values.

Select Options from the Resource Editor menu bar to display the Options menu:

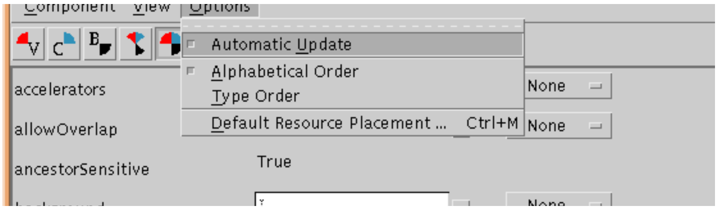


Figure 77. Options Menu

The item available from the Option menu is Automatic Update

Automatic Update

When set, updates the Resource Editor automatically each time you select a object. When Automatic Update is not set, the resources are displayed only when you click the Update button. Automatic Update is set by default.

Note: It is often more efficient to update the Resource Editor manually, for example, when you are using the Builder Xcessory editors to assign a resource value to a number of widgets.

Alphabetical Order

Displays the Resource Editor resource list in alphabetical order. Use Alphabetical Order in conjunction with Type Order to display resources alphabetically by resource type. By default, Alphabetical Order is set.

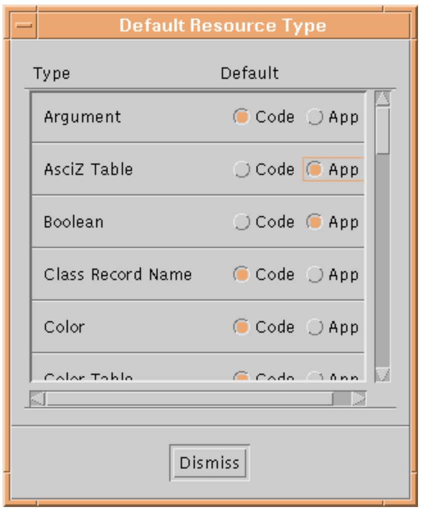
Type Order

Groups resources by resource type. For instance, createCallback, destroyCallback, focusCallback, helpCallback, mapCallback, and unmapCallback are grouped together. Use Type Order in conjunction with Alphabetical Order to display resources alphabetically within each type grouping.

Default Resource Placement (Ctrl + M)

Note: This feature is not applicable for Java.

Select Default Resource Placement from the Options menu of the Resource Editor to display the Default Resource Type dialog:



Type	Default
Argument	[✓ Code] App
AsciZ Table	Code [✓ App]
Boolean	Code [✓ App]
Class Record Name	[✓ Code] App
Color	[✓ Code] App
Color Table	[✓ Code] App

Figure 78. Default Resource Type Dialog from the Code Menu

The default controls the resource setting which a resource of the appropriate type takes when modified. For example, change the Color resource setting by clicking the App toggle button. Now change the value of a color resource for some interface object (such as background for a bulletin board widget). The modified value defaults to App, rather than Code.

Form Constraint Editor

The Form Editor provides a graphical method of accessing the constraint resources of any child of a form widget. The editor is created with the form and exists as a transparent interface to the widget. When you place widgets as children of a form, the child widget's topAttachment, rightAttachment, bottomAttachment, and leftAttachment resources are represented by attachment icons on the widget's top, right, bottom, and left edges, respectively. For example, in the following figure, the right push button has the following attachments:

- topAttachment = XmATTACH_FORM
- rightAttachment = XmATTACH_NONE
- bottomAttachment = XmATTACH_NONE
- leftAttachment = XmATTACH_WIDGET

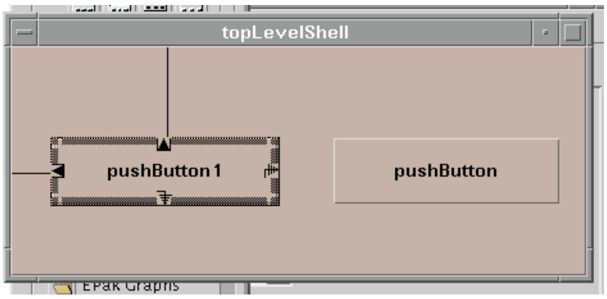


Figure 79. Form Editor

Editing attachment resources

Use the Form Editor to edit the child widget's attachment resources according to the following methods:

- Press MB3 on an attachment icon. A list of the possible values for that resource (for example, the values XmATTACH_POSITION and XmATTACH_OPPOSITE_FORM) is displayed. The toggle button for the current value is set, those for the other values are unset. Move the cursor over the desired value and release the mouse button. The attachment changes accordingly.
- Hold down Ctrl and press MB3 on an attachment icon. The current settings are displayed. Moving the cursor while holding Ctrl down allows you to change the offset or attach position.

Note: You can cancel the change by releasing the Ctrl key before the mouse button.

- Press the MB1 on an attachment and drag the cursor to the desired attachment. To attach to a widget, drag to the desired edge of the widget; to attach to the form, drag towards the edge of the form. Release the mouse button. The attachment changes accordingly.
- To set an attachment to XmATTACH_POSITION, drag MB1 from the attachment icon onto the widget itself.
- To modify an attachment's offset/position information, press Ctrl+MB3 on the attachment icon, drag to the desired location, and release Ctrl and MB3.

Attachment Symbols

In the Form Editor, the Builder Xcessory uses the following symbols to represent the various form attachment types:

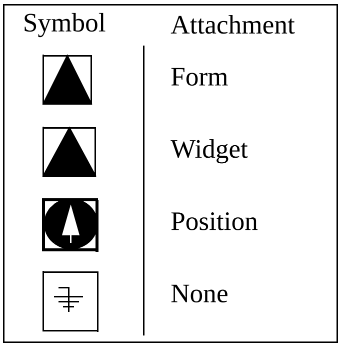


Figure 80. Form Editor Symbols

The symbols are rotated to indicate the side to which they apply:

▲	topAttachment	(form/widget)	◓	topAttachment	(self)
◀	leftAttachment	(form/widget)	◐	leftAttachment	(self)
▼	bottomAttachment	(form/widget)	◒	bottomAttachment	(self)
▶	rightAttachment	(form/widget)	◑	rightAttachment	(self)

Figure 81. Rotated Symbols

Setting attachments

For many operations on the form and its children, you do not have to use the keyboard. Mouse buttons one and three are used for many operations on the form child constraints.

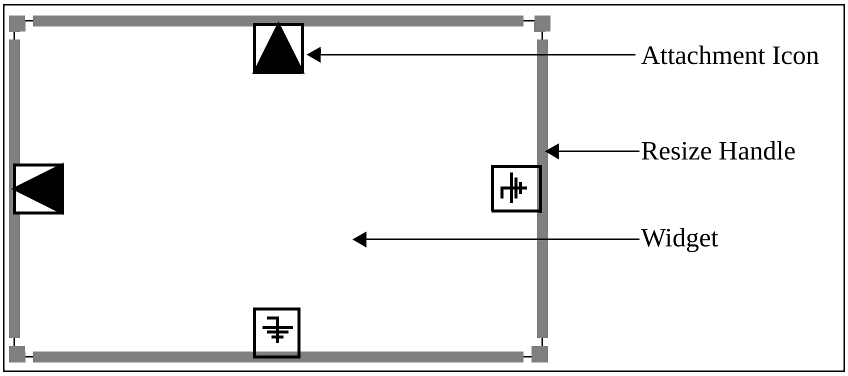


Figure 82. Setting Form Editor Attachments

All actions relating to the form attachments must be initiated on the attachment icons. When the cursor moves into the attachment icon, the cursor will change to indicate that you may perform attachment actions.

Extended Editors

Many resources have an extended editor activation push button, labeled (...), which appears to the right of the resource. Click the (...) button to invoke an editor that allows you to modify the resource.

Builder Xcessory extended editors

Builder Xcessory includes the following extended editors:

- Any Editor (see page 214)
- Application Timer Procedure Editor (see page 215)
- Application Work Procedure Editor (see page 218)
- ASCIZ Table Editor (see page 220)
- Boolean Editor (see page 221)
- Callback Editor (see page 222)
- Color Editor (see page 227)
- Color Table Editor (see page 229)
- Compound String Editor (see page 230)
- Compound String List Editor (see page 231)
- Event Handler Editor (see page 234)
- Font List Editor (see page 237)
- Integer Editor (see page 240)
- Integer Table Editor (see page 240)
- List Editor (see page 241)
- Multiline Editor (see page 241)
- One of Many Editor (see page 242)
- Pixmap Editor (see page 244)
- Translation Table Editor (see page 248)
- Widget Editor (see page 249)

The Builder Xcessory extended editors are described in *Chapter 5—Extended Editors*.

You can also add your own customized editors to the Builder Xcessory, especially in conjunction with adding user-defined widgets.

Expressions

You can use expressions to specify Integer values in the Resource Editor, the Style Editor, and the Integer Editor. Expression grammar in Builder Xcessory is as follows:

```
<expression> := <integer value> | <constant name>
    | <expression> + <expression>
    | <expression> - <expression>
    | <expression> * <expression>
    | <expression> / <expression>
    | (<expression>)
    | - <expression>
```

Expressions are evaluated from left to right. Standard order of precedence is enforced for binary operator expressions. The following conditions apply to expressions:

Expression conditions

- When an expression is entered in a resource or Style Editor text field, the expression is evaluated, and the result is assigned to the widget resource. The label to the right of the text field in the resource list is changed to Expr.
- To use an expression in the Integer Editor, select the Expr toggle. The string entered in the extended editor will be evaluated to a numeric value and that value assigned where appropriate.
- The calculated value will be re-evaluated for expressions that use constants if any of the constant values change. Any references throughout the interface will use the new value.
- If a widget or style resource value depends on an expression which includes a constant that is Cut from the Constant Manager, then the widget's resource will change from Expr to Code and its value will be set to the previously evaluated integer value.

Managers

4

Overview

Managers help you keep track of the many procedures, types, identifiers, constants, and styles you define in a large, complex interface. This chapter includes the following sections:

Browser Managers Menu

The following menu is displayed when you select Managers from the Browser menu bar:

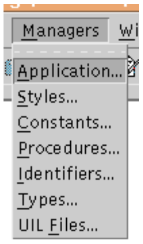


Figure 83. Browser Managers Menu

The available menu choices allow you create, edit, and delete styles, constants, procedures, identifiers, types, and files. Changes entered in the Callbacks Editor or the Creation Routine dialog are reflected in the managers, and vice-versa.

The following sections describe the Browser Managers menu options.

Application Manager

Allows you to add work procedures and timer procedures to your application. These procedures are useful if your application requires tasks beyond processing events from the user.

Displaying the Application Setup dialog

Select Application from the Browser Managers menu to display the Application Setup dialog:

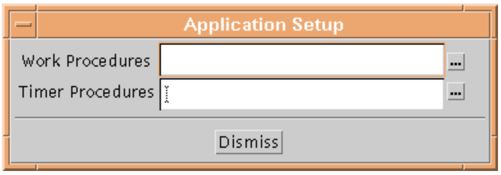


Figure 84. Application Setup Dialog

Work Procedures

A work procedure is a callback function that provides a limited form of background processing. The work procedure is invoked by the Xt Intrinsics whenever no other events are pending. That is, when X is idle (not drawing to the display or responding to user actions), the work procedure is called.

Adding a work procedure

Use a work procedure to specify a function to call when no other events are pending. Once called, a work procedure monopolizes the CPU. Therefore, a work procedure should be a function that executes quickly. Otherwise, the display stops responding to the user while the work procedure executes. For example, to perform a long calculation with work procedures, set up the calculation as an iterative process. Then, a small portion of the calculation can be done each time the work procedure is called. This approach avoids freezing the display.

Adding a work procedure by hardcoding

To add a work procedure by hardcoding, use the following function:

```
XtAppAddWorkProc(app_context, proc, client_data)
```

This function returns an ID that identifies the work procedure. The `client_data` parameter specifies application-defined data that is passed to the work procedure.

Adding a new work procedure using the Application Work Procedure Editor

To add a work procedure:

1. Click on the "..." button next to the Work Procedures input field in the Application Manager to display the Application Work Procedures Editor:

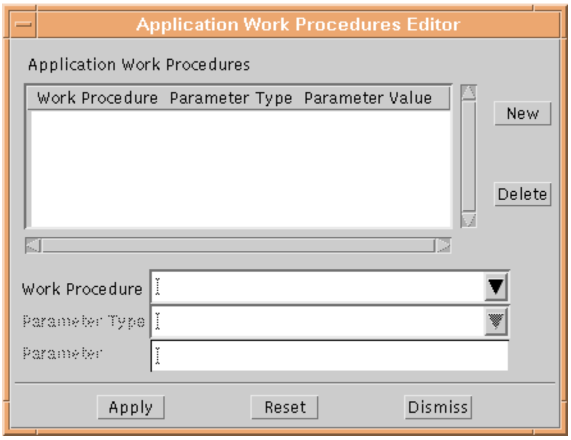


Figure 85. Work Procedures Extended Editor

2. Click on New.
3. Enter the name of the work procedure to call in the Work Procedure field.
4. Enter the parameter type in the Parameter Type field.
5. Enter the parameter (client_data) in the Parameter field
6. Click on Apply to add the work procedure.

All current work procedures are displayed in the scrolled window above these fields.

Removing a work procedure

To remove a work procedure:

1. Select the procedure from the list displayed in the Application Work Procedures Editor.
2. Click on Delete.

The procedure is removed from the list and from your application.

Timer Procedures

A timer procedure is a callback procedure that you register. A timer procedure is invoked by the Xt Intrinsics when the interval of time that you specified elapses. When a timeout event occurs, the Intrinsics invokes the timer callback procedure, and the Intrinsics automatically removes the callback. Timeout events are invoked only once.

When generating ViewKit code, Timers are implemented using the VkPeriodic class. VkPeriodic invokes the timer repeatedly until it is explicitly stopped.

Registering a timer procedure

Use a timer procedure for an event that you want to happen periodically, but not continuously. When the timer procedure is registered, Xt first waits for at least the interval you specified, and then calls the timer procedure.

Use the following function to register a timer procedure by hardcoding:

```
XtAppAddTimeOut(app_context, interval, proc,
client_data)
```

and specify the interval of time that should elapse before this function is invoked. The interval of time is specified in milliseconds. After this interval elapses, the `proc` callback function is invoked by Xt Intrinsics. The `client_data` parameter specifies client data passed to `proc`. XtAppAddTimeOut returns an ID that identifies the timer procedure.

Note: To call proc repeatedly, you must re-register the timeout within the `proc` procedure.

Application Timer Procedure Editor

To register a timer procedure using Builder Xcessory, click on the "..." button next to the Timer Procedures input field in the Application Manager to display the Application Timer Procedures Editor:

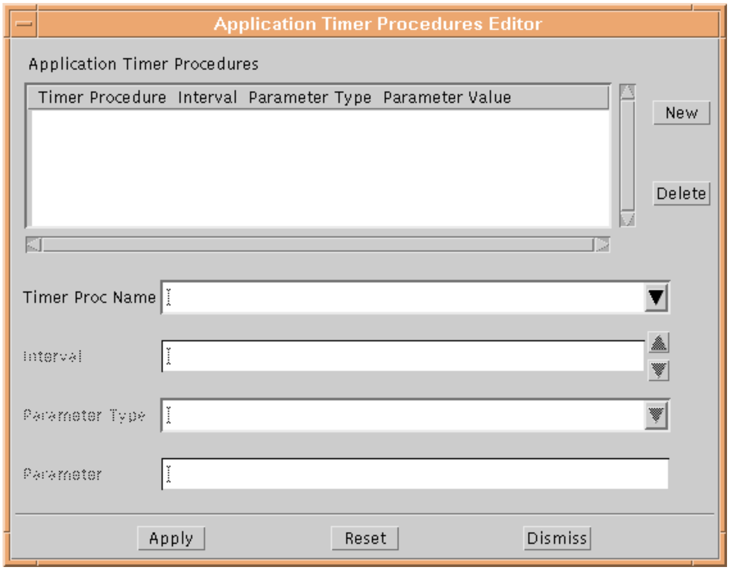

Application Timer Procedures Editor
Application Timer Procedures
Timer Procedure Interval Parameter Type Parameter Value
New
Delete
Timer Proc Name
Interval
Parameter Type
Parameter
Apply
Reset
Dismiss

Figure 86. Application Timer Procedures Editor

Registering a new timer procedure

To register a new timer procedure:

1. Click on New.
2. Enter the name of the timer procedure to call after the interval expires in the Timer Proc Name field.
3. Enter the interval in the Interval field.
4. Enter the parameter type in the Parameter Type field.
5. Enter the parameter (data) in the Parameter field.
6. Click on Apply to register the timer procedure.

Current timer procedures are displayed in the window above these fields.

Removing a timer procedure

To remove a timer procedure:

1. Select the procedure from the list displayed in the Application Timer Procedures Editor.
2. Click on Delete.

The procedure is removed from the list and from your application.

The information you enter in the input fields is used to create the XtAppAddTimeOut function in the code generated from Builder Xcessory.

Styles

A style is a set of resource values assigned to a particular name. When the style name is applied to a widget, the set of resource values is applied to that widget. Builder Xcessory styles are flexible, and provide options such as applying a given style to an entire widget tree or forcing the style resource values to override resource values previously defined for a widget.

The Builder Xcessory allows you to set resource values quickly and consistently through the application of styles. You can apply a single style, or a related family of styles, to the objects in your interface to achieve a consistent look for your application. You can also apply the same styles over a set of applications to achieve a similar look and feel.

Accessing styles

You can access styles from the following two windows:

- Style Manager

  Allows you to view and edit the hierarchy of styles that you created and to apply styles to objects.
- Style Editor

  Allows you to rename styles and apply resource values to a given style. Refer to *“Style Editor”* on page 174 for more detailed information.

Styles are defined, viewed, and manipulated using the Style Manager.

Style Manager

Allows you to view, add and edit styles, and to apply a set of resource values quickly and consistently across a number of widgets.

Select Styles from the Browser Managers menu to display the Style Manager:

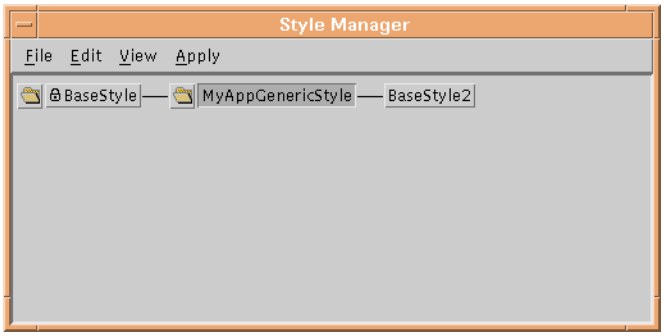


Figure 87. Style Manager

Style mouse operations

The style hierarchy is displayed within the window. Clicking MB1 on a style selects that style. The currently selected style is highlighted on the hierarchy. Double-click MB1 on a style to display the Style Editor (see *"Style Editor"* on page 174). Double-click MB3 on a style to select that style and pop up an Edit menu identical to the Style Manager Edit menu.

Lock icon

System styles are designated on the Style Manager with a "lock" icon. Although you can use the Style Editor to review the contents of these styles, you cannot edit their resource values.

L icon

Styles referenced in Palette collections are designated with an "L" icon. You can edit these styles, but you cannot delete them.

Note: Styles designated with an "L" icon remain in the Style Manager between sessions.

Style Hierarchy

The style hierarchy is displayed on the Style Manager as either an outline or a tree. Styles, like widgets, are related to one another as parents and children. Substyles inherit the resource values of their ancestors. A style includes not only the resource values defined in that style, but also any resource values defined for the style's ancestors. A value specified in the widget itself takes precedence over a value for the same resource specified in an ancestor.

Inheritance of resource values

For example, assume that you have a style hierarchy containing the BaseStyle and substyles BaseStyle1, BaseStyle2, BaseStyle3, and BaseStyle4.

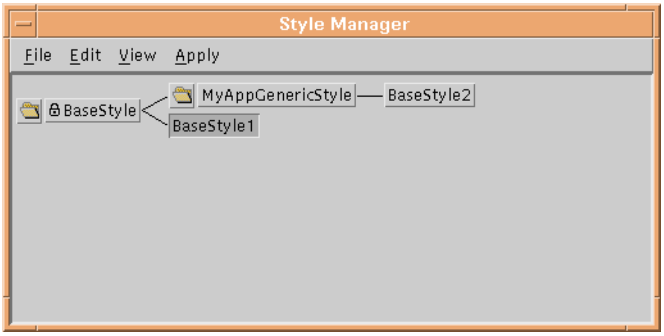


Figure 88. Sample Style Hierarchy

BaseStyle, a system style that serves as the root of the style hierarchy, is always locked. You cannot define resources for BaseStyle. In the sample style hierarchy, BaseStyle1 and BaseStyle2 are the children of BaseStyle, while BaseStyle3 is the child of BaseStyle1, and BaseStyle4 is the child of BaseStyle3.

Assume that the resource background is set as red in BaseStyle1, blue in BaseStyle2, and is undefined in BaseStyle3 and BaseStyle4. Applying BaseStyle1 to a widget applies the value red to that widget's resource background. Applying BaseStyle2 applies the value blue because the resource is defined specifically for BaseStyle2. Applying BaseStyle3 or BaseStyle4 applies the value red because, although the resource is not set specifically in either style, it is set in their ancestor, BaseStyle1.

Conflicting values in ancestors

Assume that BaseStyle4 has been applied to a widget, and you change BaseStyle3 to specify a new value, green, for background. The background of the widget changes to green because BaseStyle4 is automatically updated to reflect the resource values that are part of its ancestor style, BaseStyle3.

Note: BaseStyle4 applies the value green, rather than the value red, to the widget because BaseStyle3 is closer than BaseStyle1 to the applied style, BaseStyle4.

The following sections describe the Style Manager menu bar options.

Style Manager File Menu

Select File from the Style Manager menu bar, with the mouse or with a mnemonic, to display the following menu:

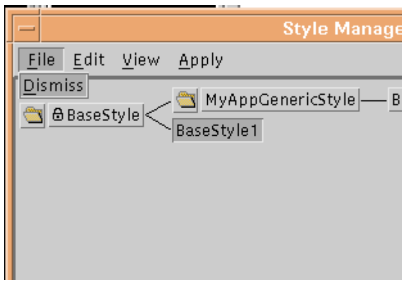


Figure 89. Style Manager File Menu

Dismiss

Allows you to dismiss the Style Manager.

Style Manager Edit Menu

Select Edit from the Style Manager menu bar, either with the mouse or with a mnemonic, to display the Style Manager Edit menu

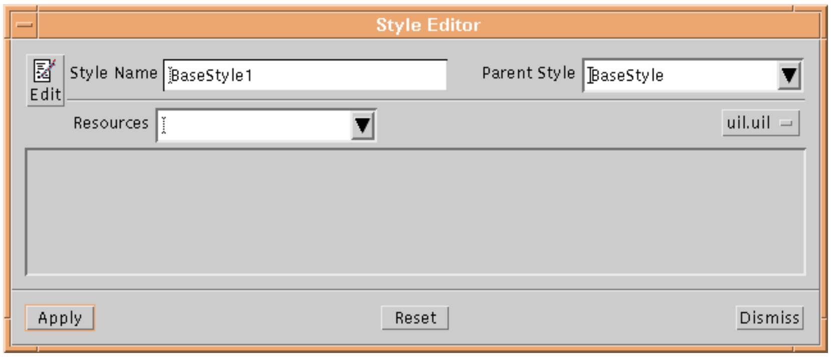


Figure 90. Style Manager Edit Menu

Note: When you hold down MB3 over a style on the Style Manager, the style is selected and a pop-up dialog identical to the Edit menu is displayed. If you click MB3 over a style on the Style Manager, the pop-up menu is displayed with the last selected menu item as the default.

The following sections describe the options available from the Edit menu.

Cut

Removes the currently selected style and its descendants from the style hierarchy and places them in the style paste buffer, overwriting any existing contents of the buffer. Use Cut in conjunction with Paste to move a style and its descendants to a new location within the style hierarchy. If you do not select Paste before another Cut or Copy operation, the Cut styles will be permanently removed.

Using drag and drop to cut and paste

You may also use drag and drop to Cut and Paste objects in the Style Manager. Dropping style A on top of style B will Cut style A and all of its descendants from the Style Manager and Paste them as descendants of style B.

Copy

Places a copy of the currently selected style and its descendants in the style paste buffer, overwriting any existing contents of the buffer. Use Copy in conjunction with Paste to copy a style and its descendants to a new location within the style hierarchy.

Using drag and drop to copy and paste

You can use drag and drop to Copy and Paste styles in the style hierarchy. Dropping style A on top of style B while depressing the Ctrl key will Copy style A and all of its descendants and Paste a copy of them as descendants of style B.

Paste

Reparents the contents of the style paste buffer as descendants of the currently selected style. Use Paste in conjunction with Copy to copy a set of styles, or with Cut to move a set of styles within the style hierarchy. Paste does not clear the style paste buffer, so you may perform multiple Paste operations.

Create Substyle

Creates a child of the currently selected style. Initially, no resource values are specified for the substyle, so it is functionally identical to its parent.

Edit Style

Displays the Style Editor for the currently selected style.Refer to *“Style Editor”* on page 174 for more detailed information about the Style Editor.

Note: You can also display the Style Editor by double-clicking on a style.

Style Manager View Menu

Allows you to display the style hierarchy as a tree or outline. Select View from the Style Manager menu bar, either with the mouse or with a mnemonic, to display the Style Manager View menu:

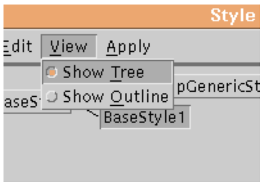


Figure 91. Style Manager View Menu

The following sections describe the View menu options.

Show Tree

Displays the style hierarchy in tree form. The display window has horizontal and vertical scrollbars. Within the window, children appear to the right of their parents, connected by lines. A toggle to the left of each parent displays either an open or closed folder. If no folder appears to the left of a style, then that style has no children. When a parent’s folder is closed, its descendants are not displayed. Clicking on a parent’s closed folder expands the display to include all of the parent's children, while subsequent generations remain hidden. Clicking on a parent's open folder hides all of the parent's descendants. By default, the style hierarchy is displayed as a tree.

Show Outline

Displays the style hierarchy in outline form. The display window has horizontal and vertical scrollbars. Within the window, successive generations appear under their parent, indented from the left margin. Siblings are indented to the same level. A toggle to the left of each parent displays as either an open or closed folder. If no folder appears to the left of a style, then that style has no children. When a parent's folder is closed, its descendants are not displayed. Clicking on a parent's closed folder expands the display to include all of the parent's children, while subsequent generations remain hidden. Clicking on a parent's open folder hides all of the parent's descendants.

Style Manager Apply Menu

Allows you to apply the currently selected style to various combinations of objects within your interface. Select Apply from the Style Manager menu bar, either with the mouse or with a mnemonic, to display the Style Manager Apply menu:

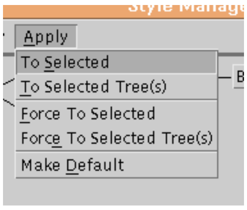


Figure 92. Style Manager Apply Menu

Note: All of the following items (with the exception of Make Default) can be applied either to an individual widget instance or to a group of widget instances. (Make Default can be applied to a single widget instance only.)

The following sections describe the Apply menu options.

To Selected

Apply style to currently selected object instance or selected group of instances.

To Selected Tree(s)

Apply style to currently selected instance and each of its descendants. If multiple instances are selected, this applies the style to each instance and its descendants.

Force To Selected	Apply style to currently selected instance, overriding any resource values previously defined for the object. You can also apply a style to a selected group of instances and override any previously-defined resource values for each instance. You can also apply a style to an object and override its resource values using drag and drop. Depress MB2 over the style, drag over the object, and release the mouse button.
Force to Selected Tree(s)	Apply style to currently selected instance and each of its descendants, overriding any conflicting resource values defined in each object. If multiple instances are selected, the style is applied to each instance and its descendants, and any conflicting resource values are overridden in each selected instance and its descendants. You can also apply a style to an instance tree and override the resource values of each object in the tree using drag and drop. Depress MB2 over the style, drag over the object and, while pressing the Ctrl key, release the mouse button.
Make Default	Make the currently selected style the default. All subsequently created objects take this style until another style is explicitly applied to them. BaseStyle is the default value. In each case, the style currently selected on the Style Manager is applied.

Style Editor

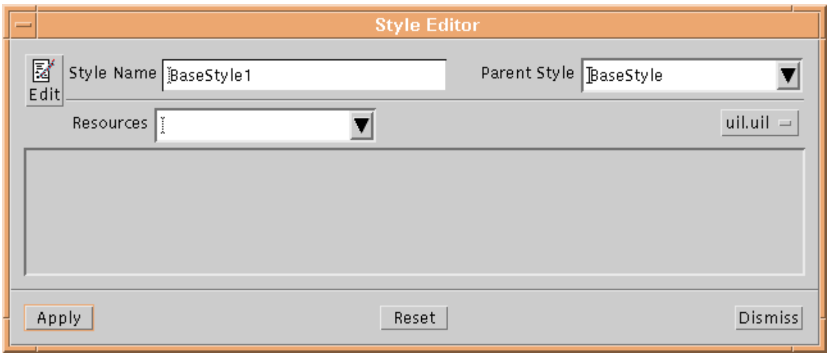
The Style Editor allows you to assign a set of resource values to a style name. Use one of the following methods to display the Style Editor:

Displaying the Style Editor

- Double-click MB1 on a style in the Style Manager.
- Select Edit Style from the Style Manager Edit menu.

Style Editor Example

Select Styles from the Browser Managers menu to display the Style Manager (if it is not already displayed). With BaseStyle as the currently selected style, select Create Substyle from the Style Manager Edit menu to create a new style. The new style, BaseStyle1, is created as the child of BaseStyle. Double-click MB1 on BaseStyle1 to display the Style Editor:



Style Name BaseStyle1
Parent Style BaseStyle
Resources
uil.uil

Figure 93. Style Editor

The following sections describe the Style Editor options.

Style Name

Displays the name of the style that is being edited.

Parent Style

Displays the name of the parent of the edited style. Clicking on the arrow button to the right of the text field displays a list of all valid styles to which this style may be parented. Selecting a different style in this field reparents the edited style. This is identical to performing a drag and drop operation on the Style Manager.

Resources

Lists the resources selected for the edited style. Click the arrow button to the right of this text field to display a combination box containing a list of all resources recognized by the Builder Xcessory.

File placement option menu

The file placement option menu (default value `uil.uil`) specifies the UIL file to which style information is written when UIL is generated. Refer to *“File Placement”* on page 198 for more detailed information.

Style Editor Drop Target

The Edit drop target is represented by a pencil and paper icon. Use the Edit drop target as follows:

- Drop a style onto the Edit drop target to update the Style Editor for that style.
- Drop a widget or its Browser widget instance name onto the Edit drop target to update the Resource Editor for that widget.

Editing styles

Add a resource to a style by entering the resource name directly into the Resources text field, or by clicking the arrow button and then selecting the resource name from the combination box.

Resources combination box

Make multiple selections from the combination box by clicking MB1 on each resource name in turn. Each selected resource is highlighted in the combination box. Deselect a resource by clicking on its highlighted name in the combination box.

Press the arrow button again to remove the combination box and display the names of all selected resources in the Resources text field.

Note: The style remains unchanged until you click on Apply at the bottom of the Style Editor.

Resource list

The resources you selected are displayed in a list in the Style Editor main display area, directly under the Resources text field. Assign values to these resources by entering the value directly into the resource text field, or by calling up any of the extended editors by clicking the (...) button to the right of the resource text field. The Builder Xcessory editors are described in *Chapter 5—Extended Editors*.

Note: The style remains unchanged until you click on Apply at the bottom of the Style Editor.

Resource placement

The resource placement of each resource is displayed to the right of the resource (refer to *“Resource Settings for a Widget Instance”* on page 135 for more detailed information). Below the resource window, the Style Editor displays the three pushbuttons, Apply, Reset, and Dismiss.

Apply

Clicking Apply on the Style Editor assigns the resource values displayed in the Resources window to the edited style.

Applying a style to a widget

To apply a style to a widget, use one of the following methods:

- Select the widget and choose the appropriate item from the Style Manager's Apply menu.
- Drag the style name from the Style Manager and drop it on the widget or the widget name on the Browser.

Applying a change to the style by clicking the Apply button on the Style Editor will immediately change the resource values of any widgets to which the style, or one of its descendants, has previously been applied.

Reset

Click Reset to read the resource values last applied to the edited style into the Style Editor.

Dismiss

Click Dismiss to remove the Style Editor window. The values in the resources window are lost unless they have been previously applied.

Saving styles

When you write out UIL, style information is included in the `uil.uil` file, or one of its include files.

Loading styles

When the Builder Xcessory attempts to Open or Read the selected file, it may display a warning dialog with the message "Name conflict between loaded style and new style" The warning indicates that a style currently in the Builder Xcessory Style Manager has the same name as a different style defined for the file you are attempting to open. The warning dialog has these push buttons:

- Use Loaded

  Overwrites the style in the file being opened with the information in the style that already exists in the Builder Xcessory Style Manager.
- Rename New

  Allows you to rename the style in the new file. The Rename Style dialog is displayed. Enter the new style name in the text field. Click OK to rename the style and remove the Rename Style dialog. If the new style name also conflicts with an existing style name, the Rename Style dialog is displayed again.

Enforced system styles

Builder Xcessory desensitizes all Resource Editor resources to which a system style is currently applied. System styles are created in the same manner as normal styles, but are written to the system file. These styles are denoted by a lock icon on the Style Hierarchy display, indicating that you cannot edit these styles. System styles allow the user to force consistent styles across a range of resources.

Adding system styles

Styles specified in `{BX}/xcessory/styles/styles.uil` are included on the Style Manager when you start the Builder Xcessory. You can add styles to this file with the following procedure:

Adding styles to ***`styles.uil`***

1. Select New from the Browser File menu, and clear everything including existing styles.
2. Create the styles you want to add to the Builder Xcessory.
3. Generate the file `uil.uil`.
4. Load the new uil.uil file into a text editor. For each style, two structures are specified in the uil.uil file:

```
list stylename : arguments
{
   arguments parentstyle;
   /* resource information goes here */
};
```

```
list stylenameReasons : callbacks
{
      callbacks parentstyleReasons;
      /* callback information goes here */
};
```

5. Append this to `{BX}/xcessory/styles/styles.uil`. Ignore the remainder of the `uil.uil` file.

To edit the hierarchy of styles specified in the `styles.uil` file, change the parent styles in both the resource and callbacks structure.

Constants

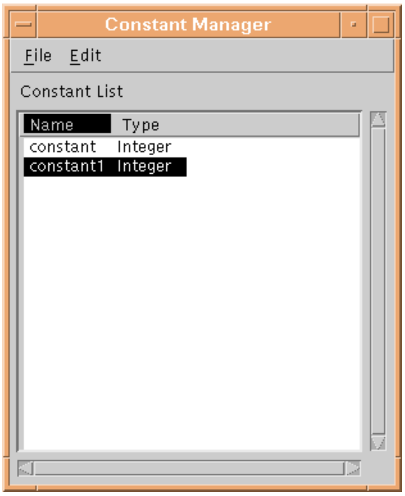
Constants are defined, viewed, and manipulated using the Constant Manager. The Builder Xcessory constants allow you to perform the following tasks:

- Assign a constant name to a value. The constant name is then referenced when you use the value in a resource. This behavior is equivalent to a style containing a single value.
- Fetch the resource value referenced by a constant at runtime using the routine call MrmFetchConstant or MrmFetchSetValue.

Note: You can use Constants of type Integer in expressions. Refer to *“Expressions”* on page 182 for more detailed information on expressions.

Constant Manager

Allows you to view constants to help you apply resource values to objects. Select Constants from the Browser Managers menu to display the Constant Manager:



Name	Type
constant	Integer
constant1	Integer

Figure 94. Constant Manager

Constant Manager File Menu

Select File from the Constant Manager menu bar, either with the mouse or with a mnemonic, to display the Constant Manager File menu:

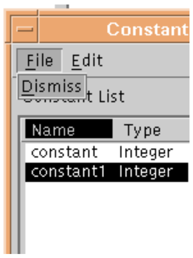


Figure 95. Constant Manager File Menu

Dismiss Allows you to dismiss the Constant Manager.

Constant Manager Edit Menu

Select Edit from the Constant Manager menu bar, either with the mouse or with a mnemonic, to display the Constant Manager Edit menu:

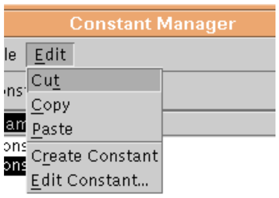


Figure 96. Constant Manager Edit Menu

The following sections describe the options available from the Edit menu.

Cut Removes the currently selected constant and puts it into the constant buffer. You are not permitted to Cut a constant which is referenced in your interface.

Copy Copies the currently selected constant into the constant buffer.

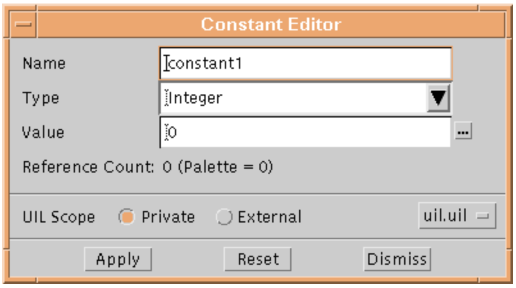
Paste Adds the contents of the constant buffer as a new constant.

Create Constant Adds a constant. The default name is constant and the default type is Integer.

Edit Constant Displays the Constant Editor for the currently selected constant. You may also display the Constant Editor by double-clicking on the constant in the Constant Manager.

Constant Editor

Select Edit Constant from the Constant Manager Edit menu to display the Constant Editor:


Constant Editor

Name: constant1
Type: Integer
Value: 0
Reference Count: 0 (Palette = 0)
UIL Scope: [✓ Private] External — uil.uil

Figure 97. Constant Editor

Name Specify the name of a constant in the Name text field.

Type Specify the type of the constant using the Type text field and its associated combination box. You may specify any of the following UIL data types:

Asciz Table	Font	Single Float
Boolean	Font List	String
Color	Integer	String Table
Color Table	Integer Table	Translation Table
Compound String	Keysym	
Float	Pixmap	

Note: Respective C data types are listed in the Type Manager.

Value Specify the resource value of the constant in the Value text field, or by using the associated extended editor (accessible by pressing the (...) button).

Reference Count

Reference Count enumerates the number of times the constant is referenced in the current interface and in the Builder Xcessory. The type of the constant may not be changed as long as this value is greater than zero. The notation Palette = n means that there are n references to the constant in collections on the Palette.

UIL Scope

Set the scope of the constant to either Private or External in the UIL output file:

- Private

  Can only be referenced in the UIL module containing the constant value declaration. The value is directly incorporated into anything in the UIL module that references the declaration.
- External

  Stored in the UID (compiled UIL) file as a named resource, and therefore can be referenced by name in other UIL files. This is the same as an "exported" UIL value.

Note: Scope affects only UIL files generated by Builder Xcessory.

Output File Option Menu

The Output File Option Menu allows you to specify the file to which the constant will be written when Builder Xcessory generates UIL. See *"File Placement"* on page 198 for more detailed information.

Using constants

You use constants in different ways, depending on whether you are generating UIL or C code.

Generating UIL

When generating UIL code, you can use constants to fetch the resource value at runtime. UIL automatically translates the data type of the constant to that of the resource value. This saves you the trouble of explicitly specifying the X, Xt, and Motif calls in the code.

For example, if you create a constant named COLOR_CONSTANT with the value "RED", the defs-c.h file will contain the line:

```
#define COLOR_CONSTANT "RED"
```

while the UIL file would contain the line:

```
value COLOR_CONSTANT = color("RED")
```

If you wanted to fetch the value in the file main-uil.c, you would include in that file the line:

```
MrmFetchConstant (file, "COLOR_CONSTANT")
```

Generating C

When generating C, you can use constants for the following tasks:

- Define named resources up front, similar to the definition of constants in the header of a program.
- Share resource definitions.

Referencing constants

Constants which have been defined may be referenced in various places in the Builder Xcessory.

Resource Editor

Enter a constant as a resource value in a resource's text field, or from the Const pop-down list off the resource placement options menu, in the Resource Editor. Refer to *Chapter 3—Resource Editor* for a detailed description of the Resource Editor.

If a widget's resource value depends on a constant that is subsequently Cut from the Constant Manager, then the widget's resource is changed from Constant to Code, and the resource value is set to the current value of the constant.

Styles

Use a constant as part of a style definition by entering the constant name in the appropriate text field on the Style Editor. Refer to *"Style Editor"* on page 174 for detailed information.

Extended Editors

Select a constant from any of the Builder Xcessory extended editors. Selecting the Constant radio button displays a combination box which contains the names of all constants of the appropriate type. Refer to *Chapter 5—Extended Editors* for a description of each of the extended editors.

Expressions

In the previous sections, a constant of type Integer may be used as part of an expression. For more information, refer to *"Expressions"* on page 160.

Example

This example describes how to apply a constant to the background resources of the widget instances in your interface:

Creating the constant

To create the constant, perform the following steps:

1. Select Constants from the Browser Managers menu.
2. Select Create Constant from the Constant Manager Edit menu.
3. Double-click on the new constant to bring up the Constant Editor.
4. In the Constant Editor, enter the name OK_LABEL in the Name text field.
5. Click the arrow button to display the Type combination box, and select Compound String. Click the (...) button to the right of the Value textfield to display the Compound String Editor.

6. Specify the value “OK” (without quotation marks) in the Compound String Editor text area. Refer to *“Compound String Editor”* on page 230 for more detailed information about the Compound String Editor.
7. Click the Apply button on the Compound String Editor to apply the string value to the constant. Dismiss the Compound String Editor. The value is entered in the Constant Editor Value text field.
8. Click the Constant Editor Apply button to create the constant OK_LABEL, displayed in the Constant Manager.

Referencing the constant

To reference the constant, perform the following steps:

1. Create a push button and update the Resource Editor.
2. Click the (...) button to the right of the labelString resource to display the Compound String Editor.
3. Click the Constant toggle to the right of Resource Placement at the bottom of the Compound String Editor. This displays a dialog with a textfield and an arrow button.
4. Click the arrow button to the right of the Constant Name text field to view all existing constants of the same type as the resource labelString (type: compoundString).
5. Select OK_LABEL and click the Apply button at the bottom of the dialog to apply the constant as the value of the push button's labelString resource. If you go back to the Constant Editor, you will notice that the Reference Count for OK_LABEL has been changed.

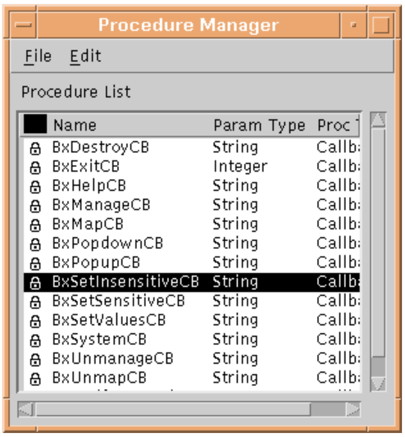
Note: If the Constant Editor is visible when the constant is applied to or deleted from a resource, then the Reference Count will be updated by clicking the Constant Editor Reset button.

Procedures

Builder Xcessory allows you to view and edit the procedures available in your interface, by using the Procedure Manager and Procedure Editor.

Procedure Manager

Allows you to define, view, and manipulate procedures available in your interface. Select Procedures from the Browser Manager menu to display the Procedure Manager:



	Name	Param Type	Proc
🔒	BxDestroyCB	String	Callb
🔒	BxExitCB	Integer	Callb
🔒	BxHelpCB	String	Callb
🔒	BxManageCB	String	Callb
🔒	BxMapCB	String	Callb
🔒	BxPopdownCB	String	Callb
🔒	BxPopupCB	String	Callb
🔒	BxSetInsensitiveCB	String	Callb
🔒	BxSetSensitiveCB	String	Callb
🔒	BxSetValuesCB	String	Callb
🔒	BxSystemCB	String	Callb
🔒	BxUnmanageCB	String	Callb
🔒	BxUnmapCB	String	Callb

Figure 98. Procedure Manager

The Procedure Manager displays the procedure name, parameter type, and procedure type of every procedure defined for your interface. The procedure name is preceded by a lock symbol if the procedure is pre-defined. Callback routines are referenced from the Callback Editor; creation routines from the Creation Routine dialog.

By default, the Procedure Manager displays all of the Builder Xcessory pre-defined callbacks. These callbacks are designated on the Procedure Manager by a lock icon, since they cannot be deleted from Builder Xcessory.

When you specify a procedure, parameter, or type in the Callback Editor or Creation Routine dialog, Builder Xcessory adds it to the Procedure Manager.

Procedure Manager File Menu

Select File from the Procedure Manager menu bar to display the Procedure Manager File menu:

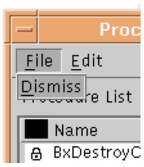


Figure 99. Procedure Manager File Menu

Dismiss Allows you to dismiss the Procedure Manager.

Procedure Manager Edit Menu

Select Edit from the Procedure Manager menu bar to display the Procedure Manager Edit menu:

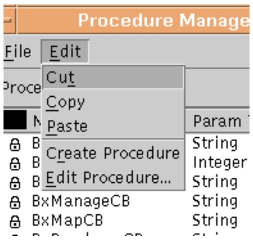


Figure 100. Procedure Manager Edit Menu

The following sections describe the options available from the Procedure Manager Edit menu.

Cut Cuts the currently selected procedure and places it in the procedure buffer. You cannot Cut a procedure referenced in your interface.

Copy Copies the currently selected procedure to the procedure buffer.

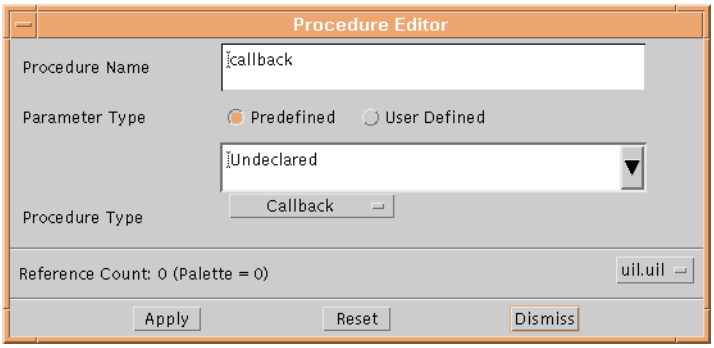
Paste Allows you to paste the contents of the procedure buffer.

Create Procedure Adds a procedure. The default procedure name is procedure, the default parameter type is Integer, and the default procedure type is Callback.

Edit Procedure Displays the Procedure Editor for the currently selected procedure.

Procedure Editor

Select Edit Procedure from the Procedure Manger Edit menu to display the Procedure Editor:



Procedure Name: callback
Parameter Type: [✓ Predefined] User Defined
Undeclared
Procedure Type: Callback
Reference Count: 0 (Palette = 0)
uil.uil

Figure 101. Procedure Editor

Procedure Name Specify the name of the procedure in the Procedure Name text field.

Parameter Type Specify whether the Parameter Type is Predefined or User Defined using the radio boxes to the right of the Type label.

- Predefined

  Defined by default when Builder Xcessory starts up. Their UIL and C/C++ equivalents are listed in the Type Manager, and appear “locked” so that they may not be cut.

- User Defined

  Appear unlocked in the Type Manager. A new type may be declared in the Procedure Editor by selecting the User Defined radio box and entering the new type in the Parameter Type text field. Builder Xcessory prompts you for confirmation and adds the new type to the Type Manager.

If the type is Undeclared, Builder Xcessory allows the user to declare the procedure parameter to be either no parameter or a parameter of any type. Otherwise, Builder Xcessory checks that the parameter passed to the procedure call is of the correct type. If the parameter type is None, Builder Xcessory will not allow the user to specify a procedure parameter when the procedure is added to a callback list or used in the Creation Routine editor. Refer to *"Creation Routine"* on page 147 for more detailed information.

When you click the arrow button to the right of the Parameter Type text field, you view all types which are either predefined or user-defined (depending on the toggle setting).

Predefined types

The following table lists predefined types:

Asciz Table	Font	Single Float
Boolean	Font List	String
Color	Integer	String Table
Color Table	Integer Table	Translation Table
Compound String	Keysym	Undeclared
Float	Pixmap	None

C data types

Equivalent C data types are listed in the Type Manager.

User Defined types

User Defined types are all other types which you have specified in the Type Manager, Procedure Manager, Callback Editor, or Creation Routine dialog.

Procedure Type

Set the Procedure Type as either Creation Routine or Callback. Creation Routine procedures may be referenced in the Creation Routine dialog. Callback procedures may be used in callback lists which are assigned as callback resource values.

A procedure's Parameter Type and Procedure Type fields may not be changed if the procedure's Reference Count is greater than zero. To decrement the count to zero, remove all references to the procedure from the current interface.

Reference Count

The Reference Count enumerates the number of times the procedure is referenced in the current interface.

Output File Option Menu

The Output File Option Menu allows you to specify the file to which the procedure will be written when Builder Xcessory generates UIL. See *"File Placement"* on page 198 for more detailed information.

Identifiers

Builder Xcessory identifiers are used for different purposes, depending on your target language.

Using identifiers in C

In C, use identifiers for the following:

- Parameters specified for callbacks and creation routines.
- Runtime values for resources.

In UIL, use identifiers for the following:

- To bind a UIL name to a value at run-time, rather than specifying the value in the code.
- Runtime values for resources.

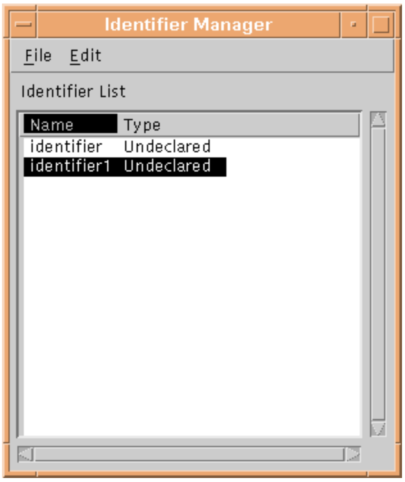
Legal characters

Legal characters include alphanumeric characters, periods, parentheses, square brackets, ampersands, asterisks, underscores, and dollar signs: ".", "(", ")", "[", "]", "&", "*", "_", "$". The characters "->" are also legal, when used together to construct an arrow. The wide variety of characters allows you to declare identifiers such as foo[4], &foo[1], *foo[1], bar(), foo::bar, structure->field, my_ident, and my$ident.

Note: Identifiers are not evaluated directly by Builder Xcessory, but are placed in output code to allow you more control over variables.

Identifier Manager

Allows you to define, view, and manipulate identifiers available in your interface. Select Identifiers from the Browser Managers menu to display the Identifier Manager:



Name	Type
identifier	Undeclared
identifier1	Undeclared

Figure 102. Identifier Manager

Using identifiers

The Identifier Manager allows you to create and edit the list of identifiers associated with a given interface. You can use these identifiers as follows:

- Parameters in callbacks accessible through the Callback Editor
- Parameters in the creation routine
- Variables which are defined at run-time

When you specify a new identifier in the Identifier Manager, Callback Editor, or Creation Routine dialog, Builder Xcessory adds the new identifier to the Identifier Manager.

Identifier Manager File Menu

Select File from the Identifier Manager menu bar (with the mouse or with a mnemonic) to display the Identifier Manager File menu:

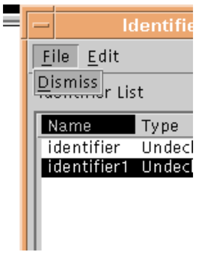


Figure 103. Identifier Manager File Menu

Dismiss Allows you to dismiss the Identifier Manager.

Identifier Manager Edit Menu

Select Edit from the Identifier Manager menu bar, with the mouse or with a mnemonic, to display the Identifier Manager Edit menu:

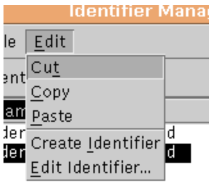


Figure 104. Identifier Manager Edit Menu

Cut Cuts the currently selected identifier and places it in the identifier buffer, provided it is not referenced in the current interface.

Copy Copies the currently selected identifier to the identifier buffer.

Paste Pastes the contents of the identifier buffer.

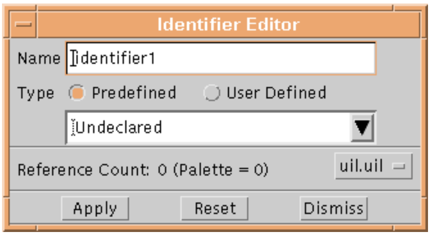
Create Identifier Creates a new identifier.

Edit Identifier

Displays the Identifier Editor for the currently selected identifier.

Identifier Editor

Select Edit Identifier from the Identifier Manager Edit menu to display the Identifier Editor:



Identifier Editor

Name: Identifier1

Type: [✓ Predefined] User Defined

Undeclared

Reference Count: 0 (Palette = 0)

uil.uil

Figure 105. Identifier Editor

Name

Specify the name of the identifier in the Name text field.

Type

Specify whether the Identifier Type is Predefined or User Defined using the toggle buttons to the right of the Type label.

- Predefined

  Defined by default when Builder Xcessory starts up. Their UIL and C/C++ equivalents are listed in the Type Manager, and appear “locked” so that they may not be cut.

- User Defined

  Appear unlocked in the Type Manager. A new type may be declared in the Identifier Editor by selecting the User Defined radio box and entering the new type in the Type text field. Builder Xcessory will prompt you for confirmation and then add the new type to the Type Manager.

Output File Option Menu

The Output File Option Menu allows you to specify the file to which the identifier will be written when Builder Xcessory generates UIL. See *“File Placement”* on page 198 for more detailed information.

Click the arrow button to the right of the Type text field to view all types which are either Predefined or User Defined (depending on the toggle setting).

Predefined types

The following table lists Predefined types:

Asciz Table	Font	Single Float
Boolean	Font List	String
Color	Integer	String Table
Color Table	Integer Table	Translation Table
Compound String	Keysym	Undeclared
Float	Pixmap	None

The equivalent C data types are listed in the Type Manager.

Note: Identifiers cannot be type None.

User-defined types

The User-defined types are all types which you have specified in the Type Manager, Procedure Manager, Callback Editor, or Creation Routine dialog.

Identifiers of type Undeclared or of any user-defined type may be used for resources that have either a type of XtPointer or no obvious type (for example, XmNcolormap or XmNscreen).

Referencing identifiers in the Resource Editor

You can enter an identifier name of the appropriate type in a resource's text field, or choose one from the Ident pop-up list on the resource placement options menu, in the Resource Editor. See *Chapter 3—Resource Editor* for more detailed information on the Resource Editor.

Note: Identifiers are not evaluated within the Builder Xcessory. The widget with a modified resource value retains its last resource value within the Builder Xcessory.

The identifier name is output as the resource value in the Builder Xcessory's generated code. You are responsible for assigning a valid value to the identifier elsewhere in your application code before your application attempts to evaluate the identifier at run-time.

When an identifier is referenced in the Builder Xcessory, you cannot cut the identifier from the Identifier Manager.

Referencing identifiers in styles

You can use an identifier as part of a style definition by entering the identifier name in the appropriate text field on the Style Editor. The identifier will remain un-evaluated. No value will be assigned to the widget resources affected by the style within the Builder Xcessory, but output code will reflect the correct value. See *"Styles"* on page 166 for more detailed information.

Referencing identifiers in Extended Editors

You can select an identifier from any of the Builder Xcessory's extended editors. Selecting the Identifier radio button displays a combination box which contains the names of all identifiers of the appropriate type. Refer to *Chapter 5—Extended Editors* for more detailed information about extended editors.

Example

This example describes how to apply an identifier to the value resource of a text widget instance in your interface.

Creating the identifier

To create the identifier, perform the following steps:

1. Select Identifiers from the Browser Managers menu.
2. Select Create Identifier from the Identifier Manager Edit menu.
3. Double-click on the new identifier to bring up the Identifier Editor.
4. In the Identifier Editor, change the name of the identifier to “my_value” (without quotation marks).
5. Change the type of the identifier to the predefined type Compound String.
6. Click the Apply button on the Identifier Editor to update the identifier. Notice that the changes are displayed in the Identifier Manager.

Create a text widget and update the Resource Editor.

Setting the value resource to the identifier

Set the value resource of this widget to the identifier in one of the following ways:

- Click on the resource placement options menu to the right of the value resource text field, and select my_value.
- Use the following procedure:

1. Click the (...) button to the right of the value resource to display the Compound String Editor.
2. Click the Ident toggle to the right of Resource Placement at the bottom of the Compound String Editor. This updates the dialog to contain a text field and an arrow button.
3. Click the arrow button to the right of the Identifier Name text field to view all existing identifiers. Select my_value and click the Apply button at the bottom of this dialog.

- Enter my_value into the value resource text field on the Resource Editor and click OK.

These methods all assign the identifier name to the text widget’s value resource in the code generated by the Builder Xcessory.

Note: The value within the Builder Xcessory does not change.

Types

Builder Xcessory has the following types:

- Predefined

  Correspond to UIL value types. The equivalent C/C++ representation for each pre-defined type is listed in the Type manager. You cannot delete pre-defined types from the Type Manager. Predefined types are designated by a lock icon in the Type Manager.

- User Defined

  Can be added to the interface by entering user-defined types in the Type Manager or in the Type or Parameter Type text field in the Callback Editor, Procedure Editor, Identifier Editor, or Creation Routine dialog. Builder Xcessory prompts you for confirmation and then adds the new type to the Type Manager.

Note: You cannot delete user-defined types from the Type Manager if they are referenced in the current interface.

Note: User-defined types must be entered in ASCII text exactly as they should be output in C/C++ files. For example, you should enter “unsigned int” in the Add Type text field to declare an unsigned integer type for use in C/C++ procedures. Legal characters include alphanumeric characters, “_”, “$”, and “*”.

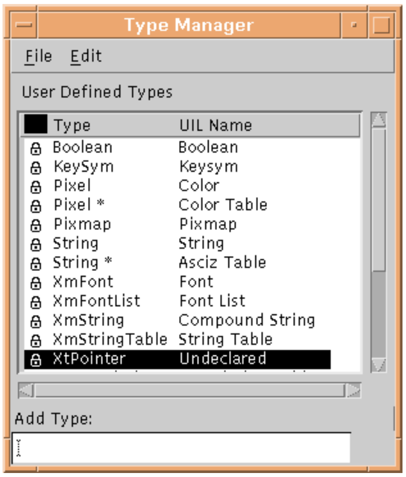
Updating predefined and user defined types

The Type Manager lists all predefined and user defined types that exist in the interface. You can update this list immediately from the following locations:

- Identifier manager
- Procedure editor
- Callback editor
- Creation routine dialog

Type Manager

Allows you to define, view, and manipulate parameter types available in your interface. Select Types from the Browser Managers menu to display the Type Manager:



Type	UIL Name
Boolean	Boolean
KeySym	Keysym
Pixel	Color
Pixel *	Color Table
Pixmap	Pixmap
String	String
String *	Asciz Table
XmFont	Font
XmFontList	Font List
XmString	Compound String
XmStringTable	String Table
XtPointer	Undeclared

Figure 106. Type Manager

Lock icons The left column in the Type Manager contains a lock icon for all Predefined types.

Type The middle column contains the representation of the type as used in the C/C++ output code.

UIL Name The right column displays the UIL equivalent for Predefined types.

Add Type To add a type, enter the name of the type in the Add Type text field at the bottom of the Type Manager and press return (or click the OK button to the right of the field). The type is added to your interface, and appears in the Type Manager. UIL names cannot be declared for User Defined types.

Type Manager File Menu

Select File from the Type Manager menu bar, either with the mouse or with a mnemonic, to display the Type Manager File menu:

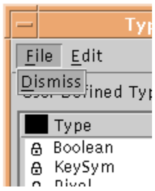


Figure 107. Type Manager File Menu

Dismiss Allows you to dismiss the Type Manager.

Type Manager Edit Menu

Select Edit from the Type Manager menu bar, either with the mouse or with a mnemonic, to display the Type Manager Edit menu:

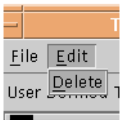


Figure 108. Type Manager Edit Menu

Delete Selecting Delete from the Type Manager Edit menu deletes the currently selected type. You will not be permitted to delete a type that is being used in your interface.

File Placement

Builder Xcessory allows you to place constants, widget resources, procedures, and styles in different UIL files. This feature is particularly useful when developing internationalized applications, because you can save language-specific information into separate files.

UIL File Manager

Allows you to view and manipulate files used for output file placement and to create and edit a hierarchy of UIL output files.

Note: With the UIL File Manager, you can use the UIL files in conjunction with Builder Xcessory File Placement to save different parts of your interface in different UIL files. This is especially useful when you are localizing an internationalized application.

Icons appear to the left of a given file name if that file is Read/Only and/or modified. Select UIL Files from the Browser Managers menu to display the UIL File Manager:

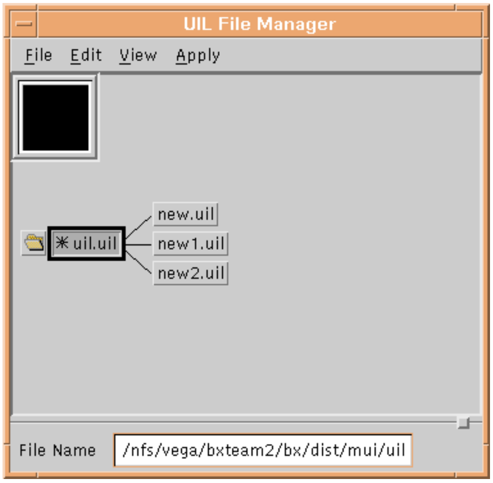


Figure 109. UIL File Manager

Read-Only UIL files Read-Only files are preceded by a lock icon. Builder Xcessory does not allow you to Save to a Read/Only file. You can change the File Name field (as described below) to write the file to a different path or file name that is not Read/Only.

Modified UIL files The names of files modified since the last Save or Save As operation are preceded by a star icon. A Save or Save As operation clears the icon.

UIL File Manager File Menu

Select File from the UIL File Manager menu bar, either with the mouse or with a mnemonic, to display the following menu:

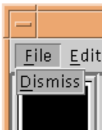


Figure 110. UIL File Manager File Menu

Dismiss Selecting Dismiss from the UIL File Manager File menu dismisses the UIL File Manager.

UIL File Manager Edit Menu

Select Edit from the UIL File Manager menu bar, either with the mouse or with a mnemonic, to display the following menu:

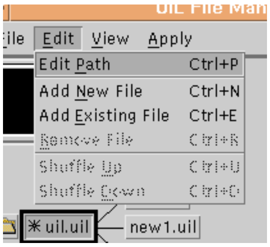
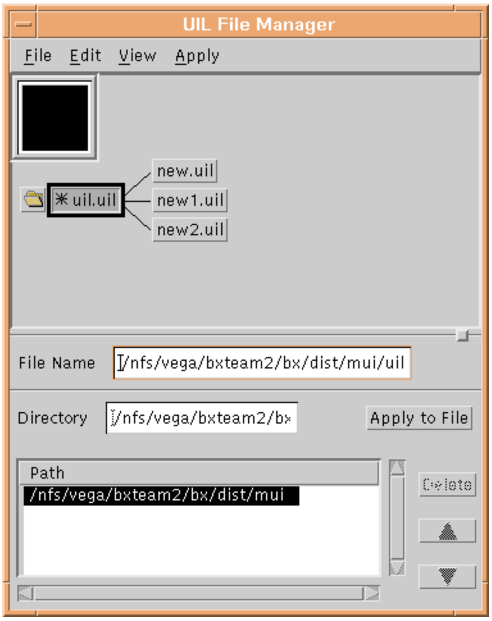


Figure 111. UIL File Manager Edit Menu

The following sections describe the UIL File Manager Edit menu options.

Edit Path (Ctrl+P) Displays the Path, in which the paths of all files in the Include File Hierarchy are displayed.



File Name /nfs/vega/bxteam2/bx/dist/mui/uil

Directory /nfs/vega/bxteam2/bx

Path
/nfs/vega/bxteam2/bx/dist/mui

Figure 112. Edit Path

To change a file path, click on the file in the hierarchy, type the new path in the UIL File Manager Directory text field, and press return or click the Apply to File button.

This applies the currently selected (highlighted) path to the selected file. This feature provides an effective method to easily move files to different directories. For example, a Read/Only file can be made Read/Write by moving it to another directory.

Add New File (Ctrl+N)

Creates a new output file, referenced in an include statement in the file currently selected in the Include File Hierarchy.

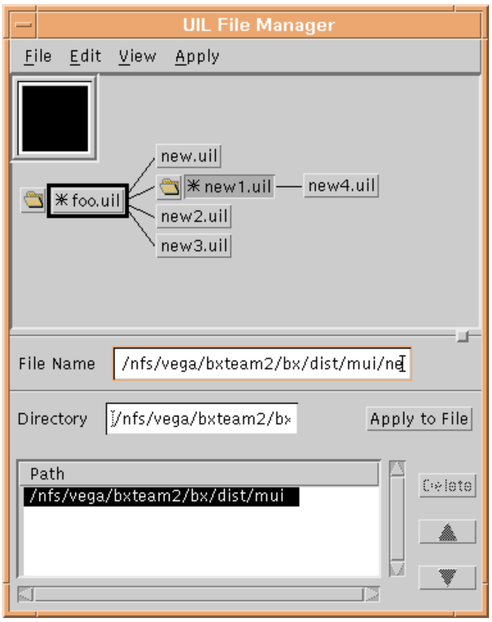


Figure 113. Add New File

File Name and Directory

To change the directory of the currently selected file to a new directory, edit the file name or the Directory text field and press return. If you change the value in the Directory text field, all files in the old directory will move to the new directory.

Path

To change the directory of the currently selected file to another directory in the Path, select the appropriate entry under Path and press Apply to File.

In each case, the File Name text field will be updated for the currently selected file.

Add Existing File (Ctrl+E)

Displays a file selection box. Select the name of an existing file in this dialog and press return. The file will be referenced in an include statement in the UIL file currently selected in the Include File Hierarchy.

Note: Existing files must not have UIL module or end module statements.

Remove File (Ctrl+R) Removes the currently selected file as well as all of its descendants in the Include File Hierarchy. You are permitted to delete a file that is used in your interface. The messages in the Browser Message window will indicate any references to the file or its descendents.

Shuffle Up (Ctrl +U) Allows you to manipulate the order in which the UIL files are included.

Shuffle Down (Ctrl +D) Allows you to manipulate the order in which the UIL files are included.

UIL File Manager View Menu

Select View from the UIL File Manager menu bar, either with the mouse or with a mnemonic, to display the UIL File Manager View menu:

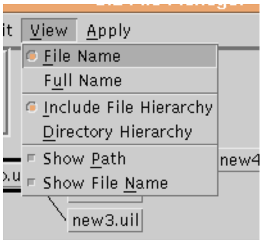
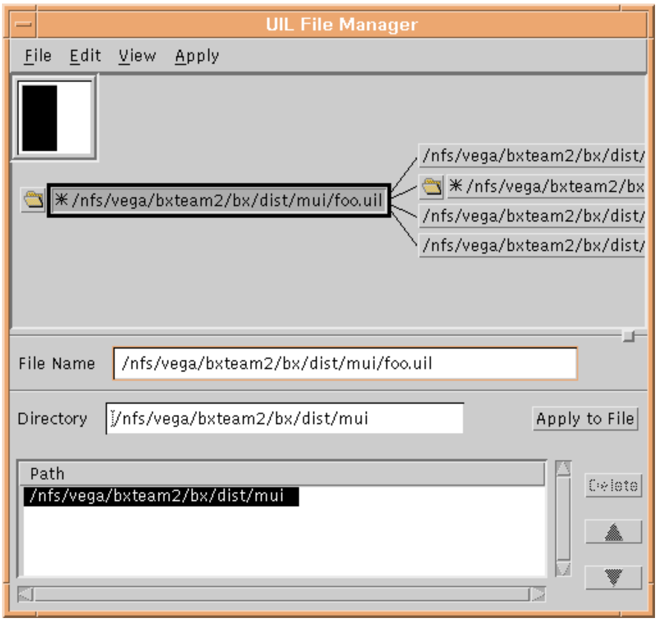


Figure 114. UIL File Manager View Menu

File Name Displays the name of each file in the Include File Hierarchy.

Full Name Displays the full path name of each file in the Include File Hierarchy.


UIL File Manager
File Edit View Apply
/nfs/vega/bxteam2/bx/dist/
✳ /nfs/vega/bxteam2/bx
/nfs/vega/bxteam2/bx/dist/
/nfs/vega/bxteam2/bx/dist/
Apply to File
Delete

✳ /nfs/vega/bxteam2/bx/dist/mui/foo.uil

File Name /nfs/vega/bxteam2/bx/dist/mui/foo.uil

Directory /nfs/vega/bxteam2/bx/dist/mui

Path
/nfs/vega/bxteam2/bx/dist/mui

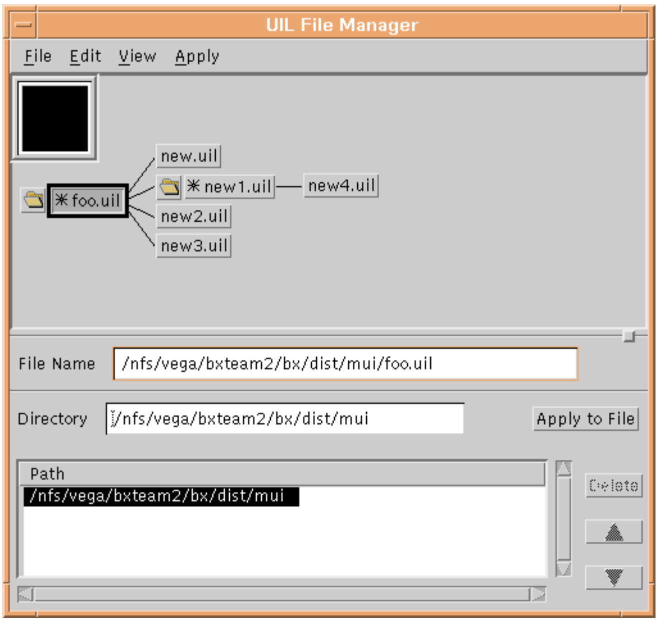
Figure 115. Full Name

Displays any logical names used when you enter a file name. Allows you to simultaneously view the data contained in the Include File Hierarchy and Directory Hierarchy view options.

Note: Full Name is insensitive when the Directory Hierarchy is selected.

Include File Hierarchy

Allows you to view the Include File Hierarchy:



File Name /nfs/vega/bxteam2/bx/dist/mui/foo.uil

Directory /nfs/vega/bxteam2/bx/dist/mui

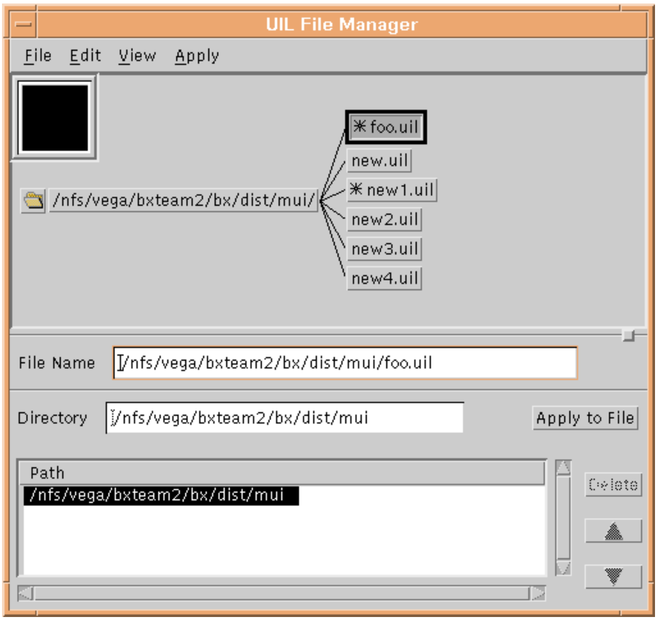
/nfs/vega/bxteam2/bx/dist/mui

Figure 116. Include File Hierarchy

The Include File Hierarchy illustrates which output files are included in which other files. For example, if `file bar.uil` is referenced in an include statement in the header of `file foo.uil`, then `bar.uil` is displayed to the right of `foo.uil`, connected by a line.

Directory Hierarchy

Allows you to view the Directory Hierarchy.



File Name /nfs/vega/bxteam2/bx/dist/mui/foo.uil

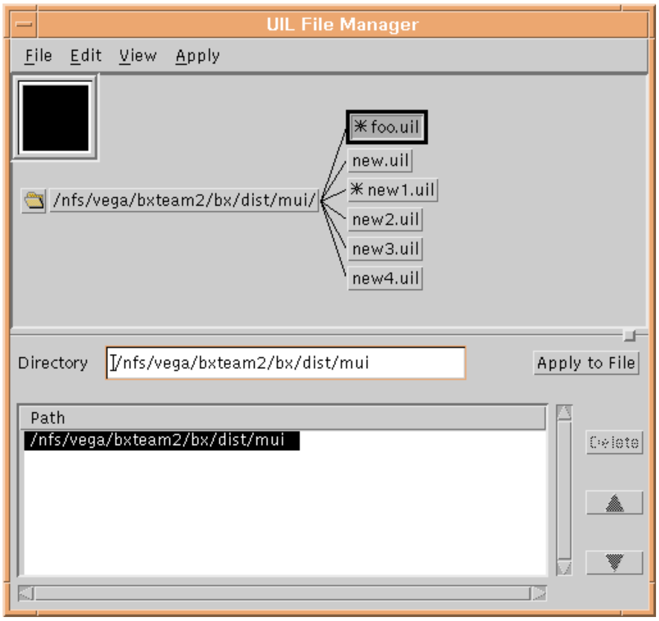
Directory /nfs/vega/bxteam2/bx/dist/mui

Path
/nfs/vega/bxteam2/bx/dist/mui

Figure 117. Directory Hierarchy

The Directory Hierarchy is a tree diagram of the directories to which the output UIL files are written. If you edit the path of a file, its relative position on the Directory Hierarchy changes. More than one “root” (left-most path) can be on this hierarchy.

Show Path Displays the Directory text field and the Path at the bottom of the UIL File Manager:



Directory: /nfs/vega/bxteam2/bx/dist/mui

Path: /nfs/vega/bxteam2/bx/dist/mui

Figure 118. Show Path

The Path consists of the directories in which the output files will be searched for, with the top-most member searched first. Shuffle the position of the selected directory path in the Path by using the up and down arrows, and delete paths that are not referenced with the Delete button.

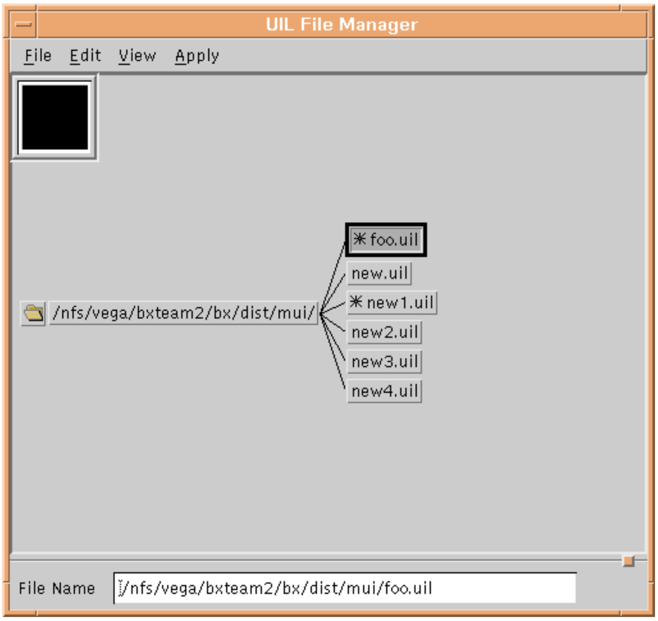
Changing a file path

To change a file path, perform the following steps:

1. Click on the file in the hierarchy.
2. Enter the new path in the UIL File Manager Path field.
3. Press return or click the Apply to File button.

Note: If the currently selected path is used by a file in the hierarchy, the Delete button is grayed out.

Show File Name Displays the File Name text field.



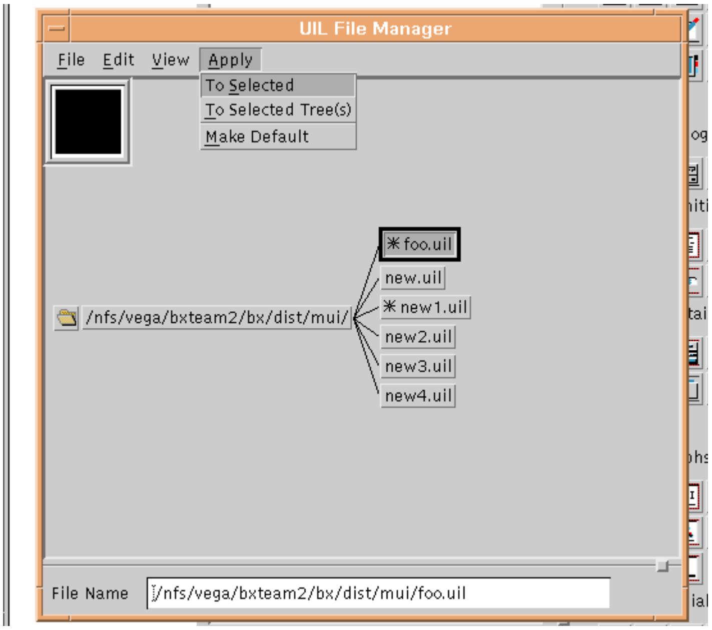
File Name /nfs/vega/bxteam2/bx/dist/mui/foo.uil

Figure 119. Show File Name

Edit the File Name text field in order to change the path or name of the currently selected file.

UIL File Manager Apply Menu

Select Apply from the UIL File Manager menu bar, either with the mouse or with a mnemonic, to display the UIL File Manager Apply menu:



File Name: /nfs/vega/bxteam2/bx/dist/mui/foo.uil

Figure 120. UIL File Manager Apply Menu

The Apply menu options allow you to apply the UIL file placement described in the following sections.

To Selected Apply UIL File Manager settings to the currently selected object(s).

To Selected Tree(s) Apply UIL File Manager settings to the currently selected object(s) and each of its descendants.

Make Default Make the currently selected file the default output file. Any subsequently created objects, styles, identifiers, and constants are placed in this file by default.

Extended Editors

5

Overview

This chapter describes the following Builder Xcessory Extended Editors:

Using Extended Editors

The Builder Xcessory extended editors allow you to modify resources and apply values for the same resource to any number of widgets or objects. Before using an editor, confirm that the currently selected object is the widget you want to set.

Resource placement options

Each extended editor has some subset of the following resource placement options:

Option	Description
None	Resets the resource value to the default.
Code	Value hardcoded and cannot be overridden by application defaults or X Resource Database.
App	Value written to a defaults file and can be overridden by the X Resource Database. **Note:** This setting is not available for Java.
Const	Takes a constant as its value. This setting displays a list in which every existing constant of the appropriate type is displayed. Value is hardcoded and cannot be overridden by application defaults.
Ident	Takes an identifier from a displayed combination box as its value. Value is hardcoded and cannot be overridden by application defaults.
Expr	Takes an expression as its value. The value is hardcoded and cannot be overridden by the application defaults. See *“Expressions”* on page 160 for more detailed information.
Style	Resource is one of a set of resources in the style that has been applied to the widget. Unless the resource was set with a style, this option is not present on the option menu.
Expose	Allows you to view and modify a widget instance’s resource when the Resource Editor is updated for the class instance in Instances View. Can be used in conjunction with any other setting except None and Style.

Extended Editor Buttons

The following three buttons appear at the bottom of each extended editor:

- Apply
  Assigns the value in the editor to the currently selected widgets resource.
- Reset
  Loads the editor with the resource value in the currently selected widget.
- Dismiss
  Removes the editor.

Identifying the Resource in the Editor Title Bar

The resource that you are setting or changing is identified in the title bar of the editor. As long as the editor for this resource is displayed, you cannot type a value into the resource field on the Resource Editor.

Updating the Resource Editor Automatically

If you select a new widget instance and press Apply, the Resource Editor sets the value for the new instance.

For example, assume you used the Color Editor to set the background of one widget to red. You can then select another widget, click Apply on the Color Editor, and set the background of that widget to red.

Note: You must update the Resource Editor each time you select a different widget.

Displaying Editor Title Bars

WmTitleBar

The Builder Xcessory uses the application resource useWmTitleBar, which takes a boolean value and is True by default. When the value is False, a label is displayed across the top of the extended editors with the same information that is in the window manager's title bar.

Note: This is useful for users who set up their window managers without title bars on transient windows. If the user sets this resource to False, and allows the window manager to set up title bars on transient windows, the displayed information is redundant.

Any Editor

Allows you to change resources of the WML data type "Any". In particular, this allows you to provide a value to the userData resource on OSF/Motif widgets.

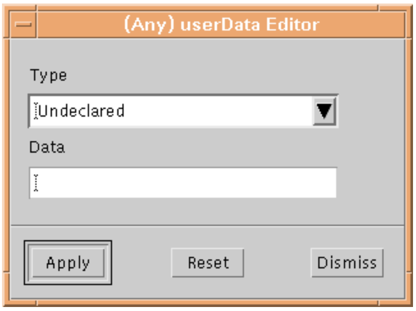


Figure 121. Any Editor

Click the arrow button to the right of the Type text field to display all the types defined for your interface. Constants, identifiers, and actual values are legitimate data values, but expressions are not supported.

Matching parameters

When Builder Xcessory encounters a parameter in the Any Editor, it searches for a match in the following order:

- Identifier
- Constant

If no match is found, Builder Xcessory prompts you to declare the parameter as an identifier.

Application Timer Procedure Editor

Allows you to add timer procedures to your application. Timer procedures allow you to specify the specific amount of time before this procedure is called. Use a timer procedure for an event that you want to happen periodically, but not continuously.

Displaying the application timer procedure editor

Click the (...) button to the right of the Timer Procedure input field on the Application Manager to display the Application Timer Procedure Editor. Refer to *"Application Manager"* on page 162.

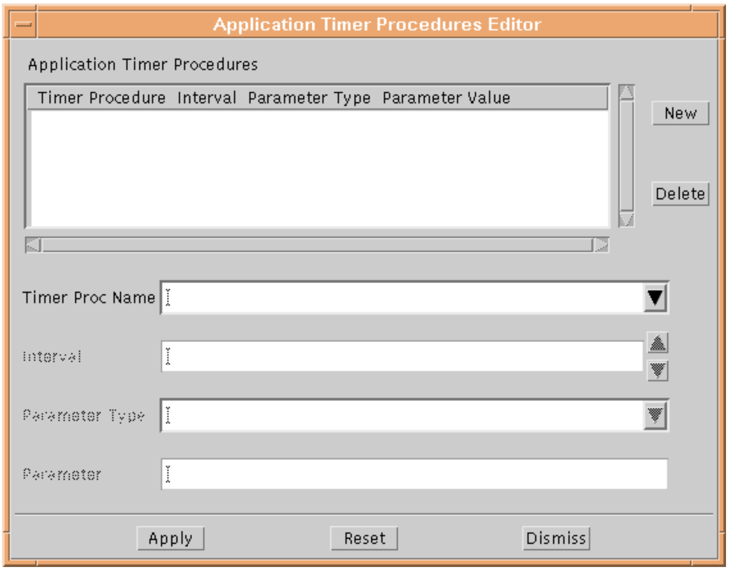

Application Timer Procedures Editor
Application Timer Procedures
Timer Procedure Interval Parameter Type Parameter Value
New
Delete
Timer Proc Name
Interval
Parameter Type
Parameter
Apply
Reset
Dismiss

Figure 122. Application Timer Procedures Editor

The display area lists the Timer Proc Name, Interval, Parameter Type, and Parameter of each timer procedure that has been added to the application.

Selecting, adding, and deleting a timer procedure

Select, add, and delete work procedures using the following methods:

- Select a timer procedure by clicking on its name in the list. All appropriate fields are updated for the selected timer procedure.
- Add a timer procedure to the list by clicking New and entering the Timer Proc Name and any necessary parameter data in the appropriate text fields.
- Delete a timer procedure by selecting it in the list and clicking the Delete button.

When changing or adding a timer procedure to the list, you can choose a procedure name from the list displayed by clicking on the arrow to the right of the Timer Proc Name label, or you can create a new procedure by typing its name in the Timer Proc Name text field. Builder Xcessory prompts you to confirm a new procedure name.

Parameter type values

New procedures are created with a default Parameter Type of Undeclared in the Application Timer Procedure Editor, indicating that parameter values of any type are accepted in the Parameter field. Other valid values include:

- Pre-defined types
- User-defined types
- None (Indicates that the timer procedure does not take any value as its client data.)

To display a list, click on the arrow to the right of the Parameter Type label in the Application Timer Procedure Editor.

Declaring additional user-defined types

You can declare additional user-defined types in the Application Timer Procedure Editor by entering the new type name in the Parameter Type field. Builder Xcessory prompts you to confirm a new user-defined type.

The Parameter Type field is insensitive if the timer procedure specified in the Timer Proc Name field is referenced by a widget or style resource in your interface. The procedure’s parameter type cannot be changed in the Application Timer Procedure Editor, unless all references to the procedure are removed from the interface.

Matching parameters from the application timer

When Builder Xcessory encounters a parameter in the Application Timer Procedure Editor, it searches for a match in the following order:

- Identifier
- Constant

If no match is found, Builder Xcessory prompts you to declare the parameter as an identifier.

Parameter values

The Parameter field accepts the following values as the client data value to pass to the callback procedure:

- Constants
- Identifiers
- Boolean
- Integer
- Floating point
- String values

Note: String values must be double-quoted.

The type of the value must correspond to the type listed in the Parameter Type field. If the text entered into the Parameter field is not a constant, identifier, or actual value, then Builder Xcessory prompts you to declare the value as an identifier of the appropriate type.

Note: Although Builder Xcessory allows you to define a series of work procedures, the X Toolkit Intrinsics execute only the most recently added procedure until that procedure is removed from the set of work procedures. Refer to your Xt documentation for more detailed information on how work procedures operate and how to write work procedures effectively.

Application Work Procedure Editor

Allows you to add work procedures to your application. This allows you to specify a function to call when there are no other X events pending.

Displaying the application manager

Click the (...) button to the right of the Work Procedure input field on the Application Manager to display Application Work Procedure Editor. Refer to *"Application Manager"* on page 162.

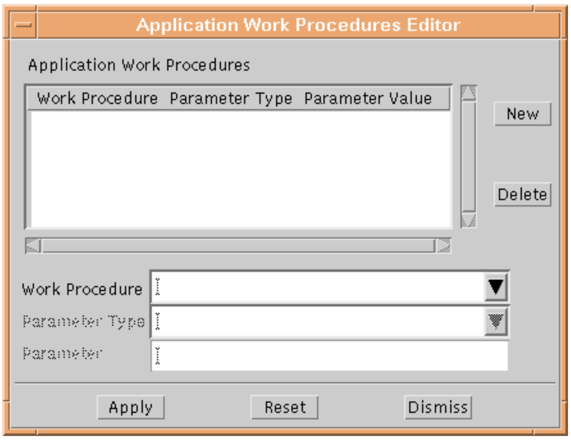


Figure 123. Application Work Procedures Editor

The display area lists the Work Procedure name, Parameter Type, and Parameter of each work procedure that has been added to the application.

Selecting, adding, and deleting work procedures

Select, add, and delete work procedures using the following methods:

- Select a work procedure by clicking on its name in the list. All appropriate fields are updated for the selected work procedure.
- Add a work procedure to the list by clicking New and entering the Work Procedure name and any necessary parameter data in the appropriate text fields.
- Delete a work procedure by selecting it in the list and clicking the Delete button.

Declaring additional user-defined types

You can declare additional user-defined types in the Application Work Procedure Editor by entering the new type name in the Parameter Type field.

Note: Builder Xcessory prompts you to confirm a new procedure name.

Parameter type values

New procedures are created with a default Parameter Type of Undeclared in the Application Work Procedure Editor, indicating that parameter values of any type are accepted in the Parameter field. Other valid values include:

- Pre-defined types
- User-defined types
- None

None indicates that the work procedure takes no value as its client data. Click on the arrow to the right of the Parameter Type label in the Application Work Procedure Editor for a list of available value.

The Parameter Type field is grayed out if the work procedure specified in the Work Procedure field is referenced by a widget or style resource in your interface. The procedure's parameter type cannot be changed in the Application Work Procedure Editor, unless you remove all references to the procedure from the interface.

Matching parameters in application work procedure editor

When Builder Xcessory encounters a parameter in the Application Work Procedure Editor, it searches for a match in the following order:

- Identifier
- Constant

If no match is found, Builder Xcessory prompts you to declare the parameter as an identifier.

Parameter values

The Parameter field accepts the following values as the client data value to pass to the callback procedure:

- Constants
- Identifiers
- Boolean
- Integer
- Floating point
- String values

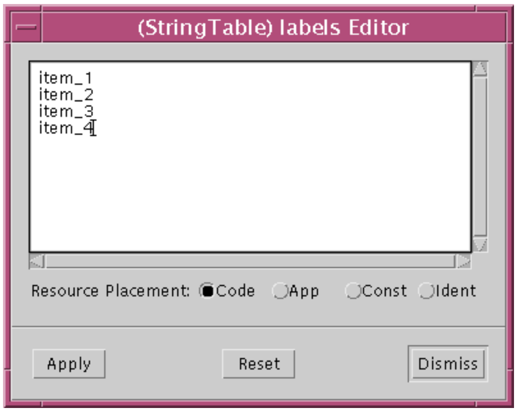
Note: String values must be double-quoted.

The type of the value must correspond to the type listed in the Parameter Type field.

If the text entered into the Parameter field is not a constant, identifier, or actual value, then Builder Xcessory prompts you to declare the value as an identifier of the appropriate type.

ASCIZ Table Editor

Allows you to specify a list of ASCIZ values for a resource.



```
item_1
item_2
item_3
item_4
```

Resource Placement: [✓ Code] App Const Ident

Figure 124. ASCIZ Table Editor

Note: You must specify each asciz value on a separate line.

For example, the following:

```
column_1

column_2

column_3

column_4
```

is output as

```
asciz_table("column_1", "column_2", "column_3",
"column_4");
```

Boolean Editor

Allows you to change the value of a Boolean resource (for example, autoUnmanage).

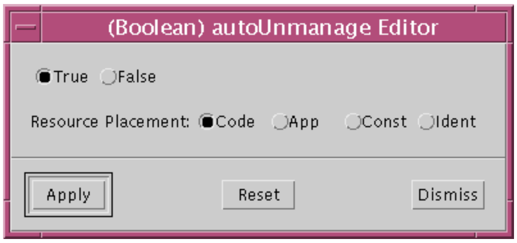


Figure 125. Boolean Editor

Set the resource value using the True and False radio buttons.

Callback Editor

Allows you to connect objects in your interface with the application. Callbacks are routines executed in response to user and program actions. The Callback Editor allows the Builder Xcessory to support callback lists (a single action results in the execution of a number of routines).

Click the (...) button to the right of any callback resource on the Resource Editor to pop up the Callback Editor.

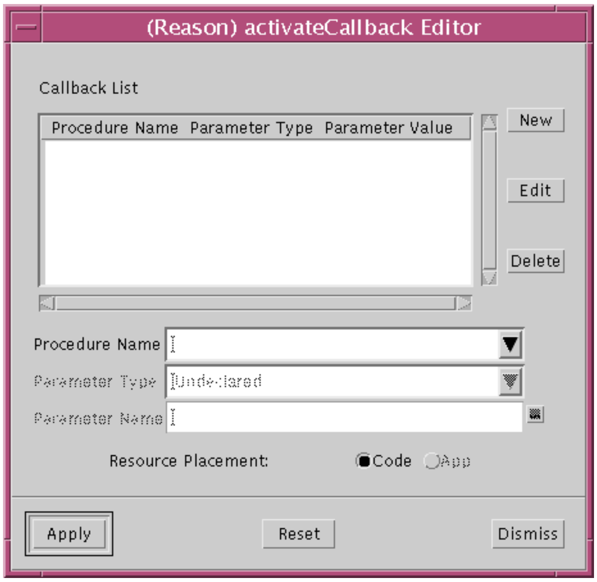


Figure 126. Callback Editor

The display area lists the Procedure Name, Parameter Type, and Parameter Value of each callback for the resource for which the Callback Editor has been called.

Selecting, adding, and deleting callbacks

Select, add, and delete work procedures using the following methods:

- Select a callback by clicking on its name in the Callback List. All appropriate fields are updated for the selected callback.
- Add a callback to the list by clicking New and entering the Procedure Name and any necessary parameter data in the appropriate text fields.
- Edit
- Delete a callback by selecting it in the list and clicking the Delete button.

Note: Builder Xcessory prompts you to confirm a new procedure name.

Parameter type values

New procedures are created with a default Parameter Type of Undeclared in the Callback Editor, indicating that parameter values of any type are accepted in the Parameter Value field. Other valid values include:

- Pre-defined types
- User-defined types
- None

None indicates that the callback procedure takes no value as its client data. All other types are listed in the Type Manager. A duplicate list is available by clicking on the arrow to the right of the Parameter Type label in the Callback Editor.

Declaring additional user-defined types

Additional user-defined types can be declared in the Callback Editor by entering the new type name in the Parameter Type field. Builder Xcessory prompts you to confirm a new user-defined type.

The Parameter Type field is grayed out if the callback specified in the Procedure Name field is referenced by a widget or style resource in your interface. The procedure's parameter type cannot be changed, in either the Callback Editor or the Procedure Editor, unless all references to the procedure are removed from the interface.

Matching parameters in the callback editor

When Builder Xcessory encounters a parameter in the Callback Editor, it searches for a match in the following order:

- Identifier
- Constant

If no match is found, Builder Xcessory prompts you to declare the parameter as an identifier.

Parameter values

The Parameter Value field accepts the following values as the client data value to pass to the callback operation:

- Constants
- Identifiers
- Boolean
- Integer
- Floating point
- String values

Note: String values must be double-quoted.

The type of the value must correspond to the type listed in the Parameter Type field.

If the text entered into the Parameter Value field is not a constant, identifier, or actual value, then Builder Xcessory prompts you to declare the value as an identifier of the appropriate type.

An extended editor for the parameter value is accessible by pressing the (...) button to the right of the Parameter Value field.

Predefined Callbacks

Click the arrow button to the right of the Callback Editor’s Procedure Name text field to display the following list of predefined callbacks (in the following table, the parameter widget list refers to a string of widget names delimited by commas: widget1, widget2, widget3...):

Predefined Callback	Parameter	Description
BxDestroyCB	Widget list	Calls XtDestroyWidget for each widget in the widget list client data parameter.
BxExitCB	Integer	Calls exit with integer parameter client data passed as the exit code.
BxHelpCB	String	Creates a dialog with a text widget and a dismiss button. The text passed as client_data is displayed in the text widget. The dialog is destroyed when the dismiss button is selected.
BxManageCB	Widget list	Calls XtManageChild for each widget in the widget list client data parameter.
BxMapCB	Widget list	Calls XtMapWidget for each widget in the widget list client data parameter. **Note:** Gadgets and other non-widget objects are not legal parameter data.
BxPopdownCB	Widget list	Calls XtPopdown for each shell in the widget list client data parameter.
BxPopupCB	Widget list	Calls XtPopup, with grab set to XtGrabNone, for each shell in the widget list client data parameter.
BxSetInsensitiveCB	Widget list	Calls XtSetSensitive with Boolean parameter set to False for each widget in the widget list client data parameter.
BxSetSensitiveCB	Widget list	Calls XtSetSensitive with Boolean parameter set to True for each widget in the widget list client data parameter.

Predefined Callback (Continued)	Parameter	Description

BxSetValuesCB	String	Passes any string of the following form:
		`"<widget_name>.<resource>=`
		`       <value>`
		`<widget_name>.<resource>=`
		`       <value>`
		`       .`
		`       .`
		`       .`
		`<widget_name>.<resource>=<value>"`
		For example:
		`"pushButton.background=red`
		`form.foreground=yellow"`
		Note: Rename widgets with unique names. Widgets assigned default names in one Builder Xcessory session (for example, pushButton, pushButton1,... pushButtonN) are not guaranteed to have the same names when the interface is saved and re-opened in a subsequent Builder Xcessory session.
		Note: Wildcard characters are not permitted in the string.

BxSystemCB	String	Passes a string from the client data parameter to the `system` system call.
BxUnmanageCB	Widget list	Calls XtUnmanageChild for each widget in the widget list client data parameter.
BxUnmapCB	Widget list	Calls XtUnmapWidget for each widget in the widget list client data parameter. **Note:** Gadgets and other non-widget objects are not legal parameter data.
BxVerifyNumericCB	None	Checks for non-numeric text input and disallows text input that contains anything other than digits. For example, this callback does not allow input of signs ("+", "-") or decimal points. Intended as the modifyVerify callback on an XmText or XmTextField widget.

Code Generation

When the callbacks-c.c file is written out, the file includes the fully commented code for the predefined callback.

Color Editor

Allows you to change any color resource.The Color Editor has two modes:

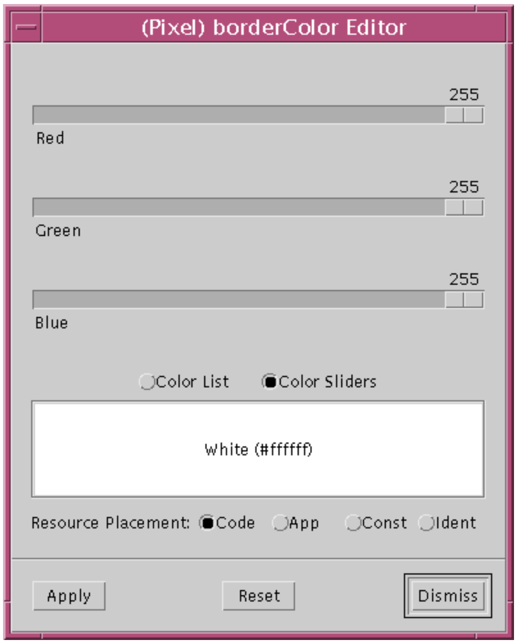
Color editor modes

- Red, green, and blue color sliders allow you to mix your own custom color.
- A color list allows you to select a named color, generated from /usr/lib/X11/rgb.txt.

If you run out of color cells on your display, the Color Editor displays a message. The correct value is assigned to the resource, and the closest existing color is used for the display.

Using the Color Sliders

Click the (...) button to the right of a color resource (for example, background) to display the Builder Xcessory Color Editor.



[Color List ✓ Color Sliders]

White (#ffffff)

Resource Placement: [✓ Code] App Const Ident

Figure 127. Color Editor Displaying Color Sliders

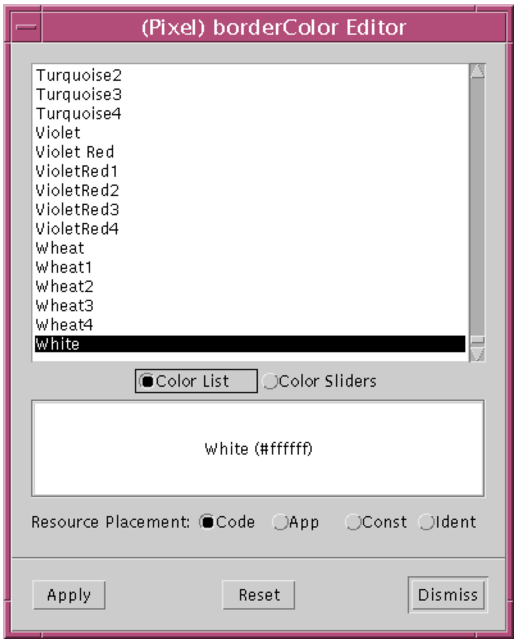
The Color Sliders option is the default. Move the red, green, and blue sliders to mix the color you wish. The color in the Color box changes as you move the sliders. When you click Apply, the selected color is recorded in the Resource Editor and represented as a six-digit hexadecimal number preceded by the pound (#) symbol.

Note: In versions of Motif before 1.1, UIL does not support hexadecimal representation. Using a custom color results in a run-time error in your compiled UIL program. You can write custom colors to app-defaults files.

If you are using Motif version 1.1 or later, you can include custom colors in your UIL files.

Using the Color List

Click the Color List option on the Color Editor to display a scrolled list of available color names:



(Pixel) borderColor Editor

Turquoise2
Turquoise3
Turquoise4
Violet
Violet Red
VioletRed1
VioletRed2
VioletRed3
VioletRed4
Wheat
Wheat1
Wheat2
Wheat3
Wheat4
White

[✓ Color List] Color Sliders

White (#ffffff)

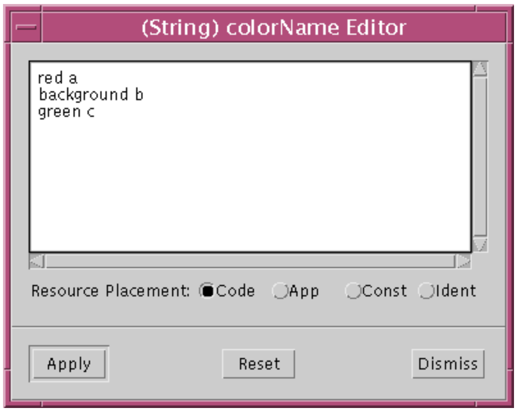
Resource Placement: [✓ Code] App Const Ident

Figure 128. Color Editor Displaying a Color List

Clicking on a color selects the color and changes the Color box. Some colors are represented in both lower-case and mixed-case. You can select either representation.

Color Table Editor

Allows you to specify a color, as well as a character that represents the color when using the icon function. The keywords background and foreground are also recognized and output appropriately.



```
red a
background b
green c
```

Resource Placement: [✓ Code] App Const Ident

Figure 129. Color Table Editor

Note: You must specify each color/character pair on a separate line. For example, the following:

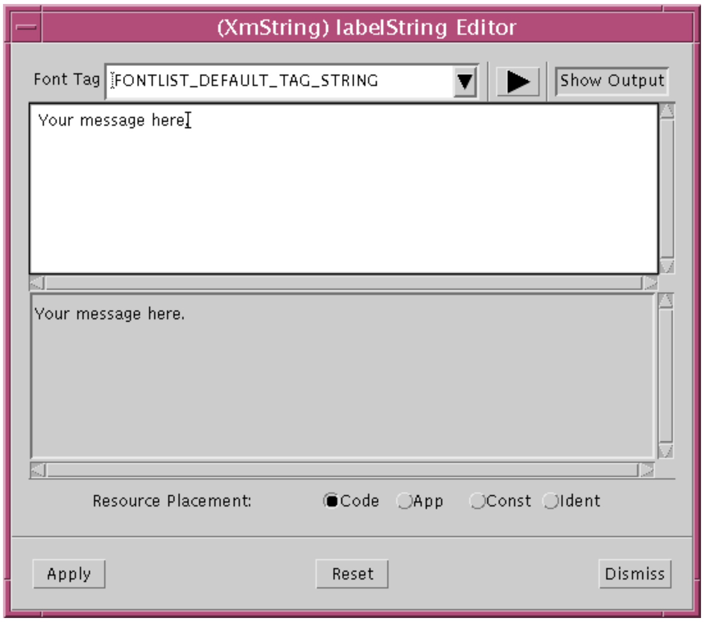
```
red a
background b
green c
```

are output as

```
color_table(color('red') = 'a',
                        background color = 'b',
                        color('green') = 'c');
```

Compound String Editor

Allows you to change a resource of type XmString (for example, labelString).



Font Tag FONTLIST_DEFAULT_TAG_STRING

Your message here

Your message here.

Resource Placement: [✓ Code] App Const Ident

Figure 130. Compound String Editor

Compound string editor components

The Compound String Editor contains the following components:

- A combination box displaying the Font Tag. The default font tag value is FONTLIST_DEFAULT_TAG_STRING. The other tags are retrieved from the associated fontList resource if you are editing a widget resource (Styles are not supported).
- An arrow button, indicating in which direction the selected text is read. The default is left-to-right (arrow points right).

Note: Vertical text is not supported.

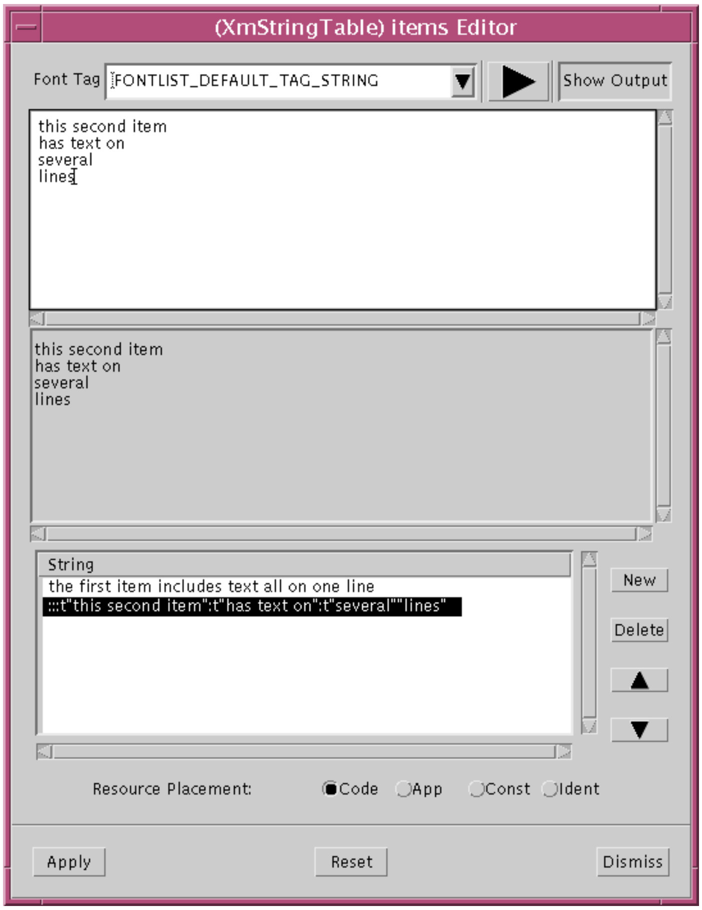
- The Show Output toggle. When the toggle is set, the compound string is displayed in a window underneath the text entry field. Any changes made in the Compound String Editor are reflected in this window before the contents are applied to the currently selected widget resource using the

OK button.

- The upper text entry field in which you edit the string and apply different font tags to different segments of the string.
- The lower text field in which the string is displayed when the Show Output toggle is set.

Compound String List Editor

Allows you to specify a list for a resource of type XmString (such as items).


(XmStringTable) items Editor

Font Tag: FONTLIST_DEFAULT_TAG_STRING

```
this second item
has text on
several
lines
```

```
this second item
has text on
several
lines
```

String
the first item includes text all on one line
:::t"this second item":t"has text on":t"several""lines"

Resource Placement: [✓ Code] App Const Ident

Figure 131. Compound String List Editor

Compound String List Editor components

The Compound String List Editor contains the following components:

- A combination box displaying the Font Tag. The default font tag value is FONTLIST_DEFAULT_TAG_STRING. The other tags are retrieved from the associated fontList resource if you are editing a widget resource (Styles are not supported).
- An arrow button, indicating in which direction the selected text is read. The default is left-to-right (arrow points right).

Note: Vertical text is not supported.

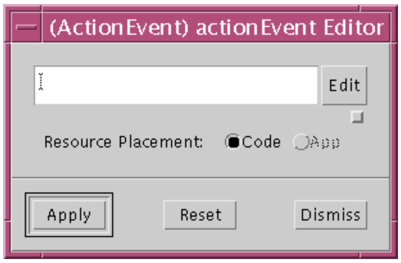
- The Show Output toggle. When the toggle is set, the compound string is displayed in a window underneath the text entry field. Any changes made in the Compound String Editor are reflected in this window before the contents are applied to the currently selected widget resource using the OK button.
- The text entry field in which you edit the string and apply different font tags to different segments of the string.
- The text field in which the string is displayed when The Show Output toggle is set.
- The String field, in which each string in the list is displayed in its string format. The New and Delete buttons to the right of this field allow you to manipulate the list of strings, while the up and down arrow buttons allow you to reorder the strings.

The Compound String Table Editor allows you to use Shift+Return to do a New operation.

Event Editor

Note: This extended editor is available in Java only.

Allows you to specify a method to call when an event occurs. Click the button to the right of the actionEvent resource in the Resource Editor to pop up the Event Editor:



Resource Placement: [✓ Code] App

Figure 132. Event Editor (Java only)

Click on the Code or App toggle of the Resource Placement field. Enter the event name in the text field and click on Apply. Click on Edit to generate code.

Event Handler Editor

Allows you to add event handlers to your application's widgets. Event handlers are attached to widgets and are accessed through the Resource Editor, much like callbacks. Event handlers are not application global. The purpose of event handlers is to let you specify functions to be called when widgets receive certain events.

Click the (...) button to the right of the eventHandler resource in the Resource Editor to pop up the Event Handler Editor:

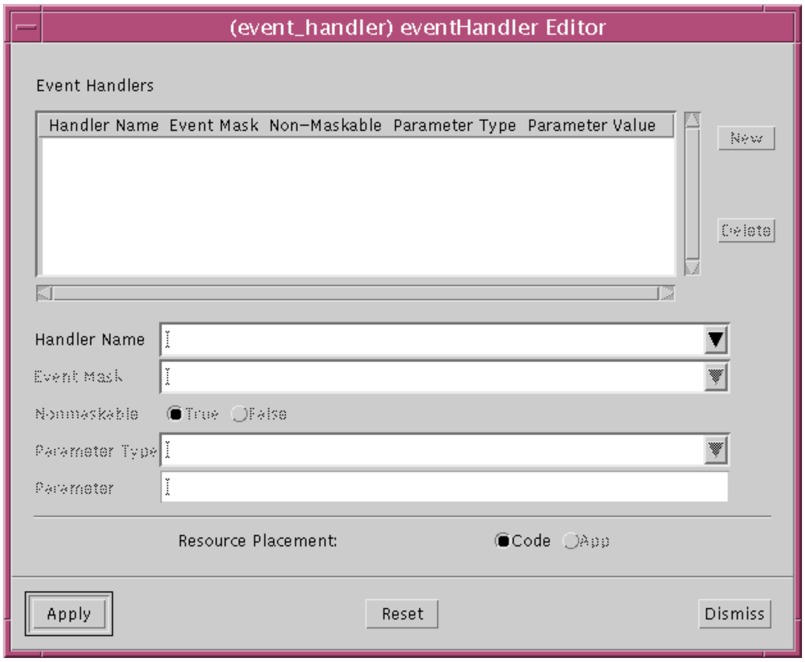


Figure 133. Event Handler Editor

The display area lists the Handler Name, Event Mask, Non-Maskable setting, Parameter Type, and Parameter of each event handler that has been added to the application.

Selecting, adding, and deleting an event handler

To select an event handler, click on its name in the list. All appropriate fields are updated for the selected event handler.

To add an event handler to the list, click New and enter the Handler Name and any necessary parameter data in the appropriate text fields.

To delete an event handler, select it from the list and click the Delete button.

When changing or adding an event handler to the list, you can choose a handler name from the list displayed by clicking on the arrow to the right of the Handler Name label, or you can create a new procedure by typing its name in the Handler Name text field. Builder Xcessory prompts you to confirm a new handler name.

When a widget gets a specified X event, the event handler is called. To specify the X event(s), you must do so in the Event Mask field. You can choose an event mask from the list displayed by clicking on the arrow to the right of the Event Mask label, or type its name in the Event Mask text field. You can choose one or more event masks, separating each mask name with a comma or a logical “or” (|).

New event handlers

New event handlers are created with a default Parameter Type of Undeclared in the Event Handler Editor, indicating that parameter values of any type are accepted in the Parameter field. Other valid values include all pre-defined types, user-defined types, and None. None indicates that the event handler takes no value as its client data. A list is available by clicking on the arrow to the right of the Parameter Type label in the Event Handler Editor.

Declaring additional user-defined types

Additional user-defined types can be declared in the Event Handler Editor by entering the new type name in the Parameter Type field. Builder Xcessory prompts you to confirm a new user-defined type.

The Parameter Type field is grayed out if the event handler specified in the Handler Name field is referenced by a widget or style resource in your interface. The procedure’s parameter type cannot be changed, in the Event Handler Editor, unless all references to the handler are removed from the interface.

Matching parameters with the event handler editor

When Builder Xcessory encounters a parameter in the Event Handler Editor, it searches for a match in the following order:

- Identifier
- Constant

If no match is found, Builder Xcessory prompts you to declare the parameter as an identifier.

The Parameter field accepts the following values as client data values to pass to the callback procedure.:

- Constants
- Identifiers
- Boolean
- Integer
- Floating point
- String

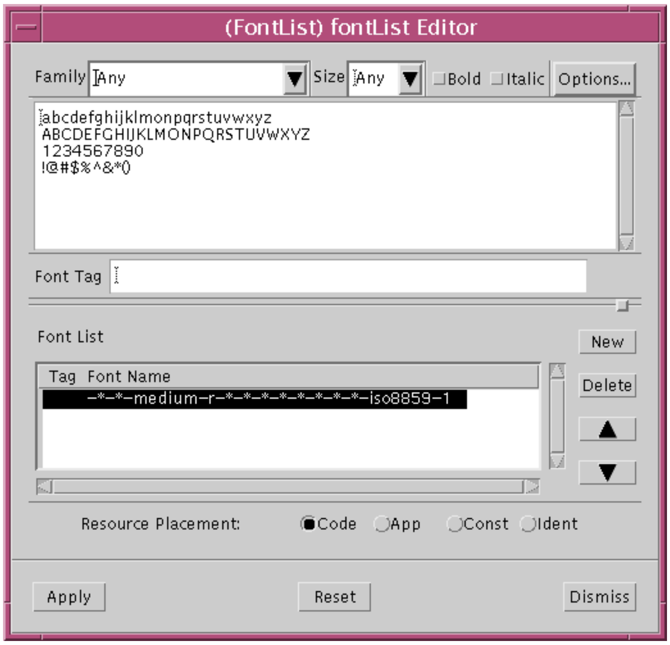
Note: String values must be double-quoted.

The type of the value must correspond to the type listed in the Parameter Type field. A list of parameters is available by clicking on the arrow to the right of the Parameter label in the Event Handler Editor.

If the text entered into the Parameter field is not a constant, identifier, or actual value, then Builder Xcessory prompts you to declare the value as an identifier of the appropriate type.

Font List Editor

Allows you to select from a list of fonts and font families, specifying style, size, slant, and weight. A window displaying sample text is automatically updated to reflect your selections. The list of fonts is generated by the Builder Xcessory from the X11 fonts available on the display on which the Builder Xcessory is running. You cannot use the Font List Editor if you want to use a font other than those present on your development system, but you can type the font name directly into the Resource Editor.



```
abcdefghijklmonpqrstuvwxyz
ABCDEFGHIJKLMONPQRSTUVWXYZ
1234567890
!@#$%^&*()
```

Tag	Font Name
	-*-*-medium-r-*-*-*-*-*-*-*-*-iso8859-1

Resource Placement: [✓ Code] App Const Ident

Figure 134. Font List Editor with Options

The Font List Editor contains the following fields and buttons:

Family A combination box that enables you to specify the font family.

Size A combination box for specifying the font size in points.

Toggle buttons Bold and Italic toggle buttons to control the weight and slant of the font.

Options button, which toggles between showing and hiding the additional options described below.

Font Tag Text field that specifies the tag to apply to currently selected font.

Font List Provides a list that enables you to apply the tag specified in the Font Tag text field to the font currently selected in the list. The New and Delete buttons to the right of this field allow you to manipulate the list of fonts, while the up and down arrow buttons allow you to reorder the list.

The Font List Editor dynamically removes choices that are inappropriate. This enables the user to freely experiment with different combinations and ensures that the selected font exists on the machine from which the Font List Editor is running.

Font list example For example, if the Times Roman 14 point font is not available, and the user selects a point size of 14, then Times Roman will not be available in the family combination box. Likewise, if the user had chosen Times Roman from the family dialog box then a size of 14 would not be shown. To have all choices available choose a size and family of Any.

Options The Font List Editor provides tremendous flexibility in font choices. Click the Options button to display an additional panel of controls. This allows the user to gain access to non-X Logical Font Description (XLFD) fonts, control the resolutions of the fonts chosen, choose from fixed or proportional fonts only, remove the use of font scaling, view non-iso8859-1 fonts, and see the XLFD name of the font which the Font List Editor is constructing.

Non-XLFD fonts

By choosing the Other Fonts toggle from the option panel, the family and size lists, as well as the bold and italic toggles, are replaced with a list of all non-XLFD fonts available on your system. This feature allows users to select non-XLFD fonts with the Font List Editor. The text field of the combination box may be edited by the user and any string entered is interpreted as a font name.

Note: XLFD names may be entered in this field. This feature allows the Font List Editor to be used to specify any font on the entire system.

Resolution control

The Font List Editor finds which of the two standard resolutions (in dots per inch—dpi) the current display is closest to and uses that as its default. To allow a wider range of choices, a user may choose to access fonts of a different resolution, or both 75 and 100 dpi resolutions.

Fixed or proportional

Some applications, such as terminal emulators, require a fixed-width font, although proportional fonts often look better. The Font List Editor allows users to limit the font choices to fixed-width or proportional, or to allow both.

Font scaling

The font scaling technology available in X11R5 uses bitmap scaling which often results in ugly fonts. To remove the scaled fonts from the list of choices, unset the Use Font Scaling toggle. You can set this value as a resource.

XLFD name display

Advanced users may desire to know which font the Font List Editor is constructing for them. By setting the Show Font Name toggle the XLFD name is shown to the user at the bottom of the Font List Editor as it is being constructed. Builder Xcessory uses wildcards when constructing a font name.

Encoding

Most English fonts have an encoding of iso8859-1. The Encoding options menu allows you to choose any of the character sets available in your interface. The character sets are loaded from your WML files.

Integer Editor

Allows you to change integer resource values (such as border width).

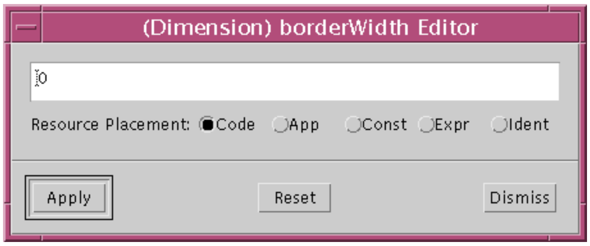

(Dimension) borderWidth Editor
0
Resource Placement: Code App Const Expr Ident
Apply
Reset
Dismiss

Figure 135. Integer Editor

Type the new integer value into the text window.

Integer Table Editor

Allows you to specify a list of integers for a resource.

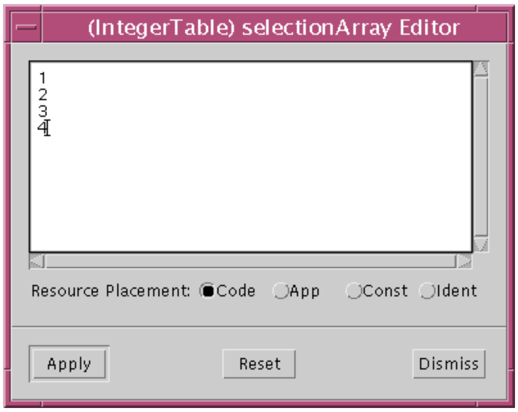

(IntegerTable) selectionArray Editor
1
2
3
4
Resource Placement: Code App Const Ident
Apply
Reset
Dismiss

Figure 136. Integer Table Editor

You must specify each integer value on a separate line. For example, value:

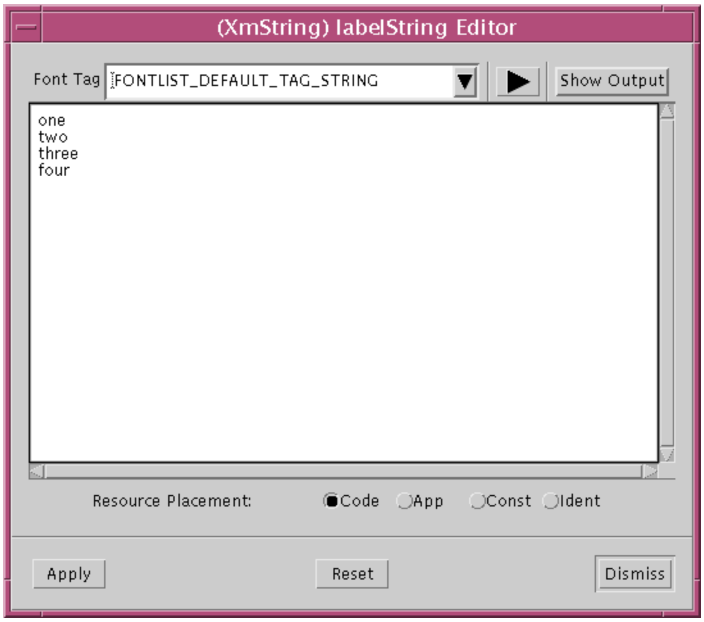
```
1
2
3
4
```

is output as

```
integer_table(1, 2, 3, 4);
```

List Editor

Allows you to create and maintain a list of strings or multibyte strings. Generally, you use the List Editor for prototypes, because the contents of a list are typically under the control of your application.



Font Tag: FONTLIST_DEFAULT_TAG_STRING

```
one
two
three
four
```

Resource Placement: [✓ Code] App Const Ident

Figure 137. List Editor

Multiline Editor

Use to enter one or more lines of text, or multibyte characters when appropriate.

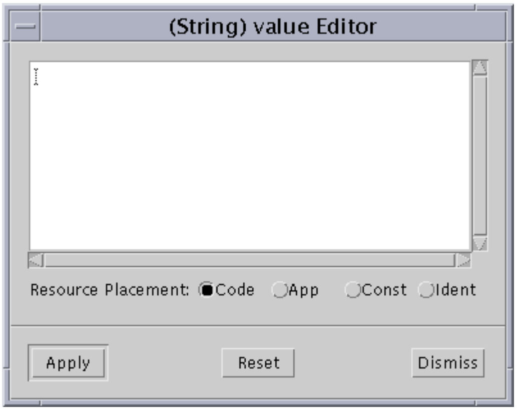


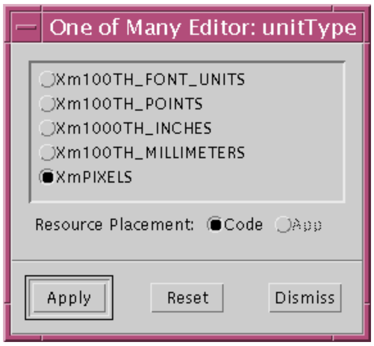
Figure 138. Multiline Editor

Type the desired text into the editor. As with all text fields in the Builder Xcessory, you can highlight text and hit the backspace or delete key to remove the text. If you highlight an area of text and begin typing, the marked text is deleted and replaced with the new text you enter.

Remember that you cannot enter a value for a resource on the Resource Editor as long as the editor for that resource is displayed. You must first Dismiss the editor.

One of Many Editor

Allows you to change a resource that has many possible values.



Xm100TH_FONT_UNITS
Xm100TH_POINTS
Xm1000TH_INCHES
Xm100TH_MILLIMETERS
[✓ XmPIXELS]

Resource Placement: [✓ Code] App

Figure 139. One of Many Editor

The window displays all of the possible choices. Click on the radio button corresponding to your desired choice.

Pixmap Editor

Along with the Pixmap Chooser, allows you to create, load, edit, and apply pixmaps. The Builder Xcessory reads pixmaps in XBM and XPM.1, XPM.2, or XPM.3 formats, and writes XPM.3.

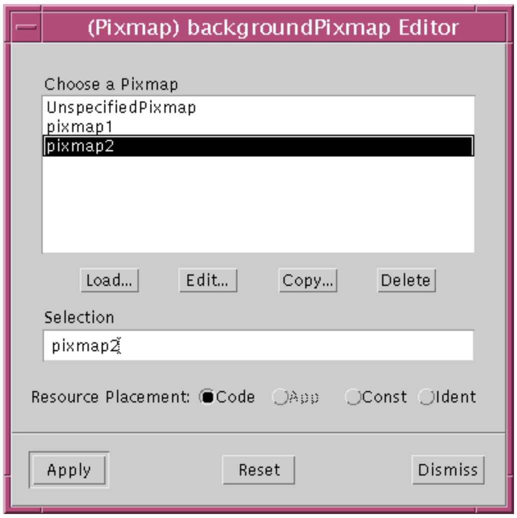
Using XPM.1 or XPM.2 pixmaps

To use XPM.1 or XPM.2 pixmaps in your interface, use one of the following methods:

- Load the pixmaps into the Builder Xcessory and write them out as XPM.3.
- Edit the code to conform to XPM.3, and insert it within the convenience routines at the top of the creation-c.c file.

Pixmap Chooser

The Pixmap Chooser is the first window displayed:



pixmap2

Resource Placement: [✓ Code] App Const Ident

Figure 140. Pixmap Chooser Window

Choose a Pixmap

The Pixmap Chooser displays a list of pixmaps in the Choose a Pixmap field. Select a pixmap from this list by clicking on its name. The default pixmap, Unspecified Pixmap, unsets the resource when selected.

Function buttons

You can perform the following Pixmap Chooser functions on an existing pixmap:

- Load reads the pixmap specified in the Selection text field into the editor.
- Edit displays the Pixmap Edit window for the currently selected pixmap.
- Copy copies the selected pixmap. You are prompted for the name of the new pixmap.
- Delete removes the selected pixmap from the Pixmap Chooser.

Editing a Pixmap

The Pixmap Editor features a Panner/Porthole combination, like that used in the Browser, which allows the user to scroll the drawing area in two dimensions.

The Pixmap Editor creates a simple color pixmap editor that allows users to generate graphics for use in an application. It supports many common functions including point, line, circle, filled circle, rectangle, filled rectangle, moving and copying areas, flood fill, and undo. The image may also be resized or zoomed by the user.

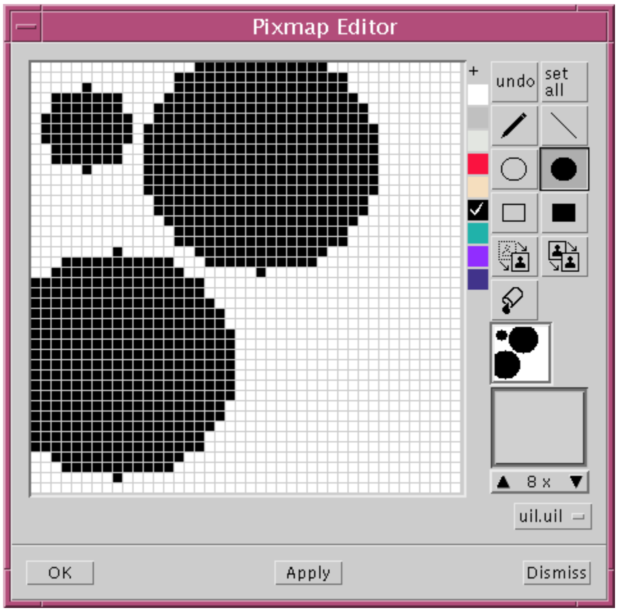


Figure 141. Pixmap Editor

The following sections describe Pixmap Editor functions:

Undo

To undo the most recent operation (only the most recent operation can be undone) select the Undo button.

Set All

To fill the entire image with a given color, and erase its current contents, select the Set All button.

Draw Point

Draws single points on the display in the current color. To draw a single point, click and release MB1. To draw multiple points, click and drag the mouse, points are drawn until the mouse button is released.

Draw Line

To draw a line, press MB1 to set one end of the line, then drag the mouse and release the mouse button to set the other end of the line.

Draw Circle

To draw a circle, press MB1 to set the center of the circle and drag to set the radius of the circle.

Filled Circle

To draw a filled circle, press MB1 to set the center of the circle and drag to set the radius of the circle.

Rectangle

To draw a rectangle, press MB1 to set one corner of the rectangle, then drag the mouse and release the mouse button to set the opposite corner.

Filled Rectangle

To draw a filled rectangle, press MB1 to set one corner of the rectangle, then drag the mouse and release MB1 to set the opposite corner.

Copy

To copy an area of the screen to another location, press MB1 to set the upper left-corner and drag out a rectangle encompassing the area to copy. Release MB1 and move the selected area to the upper-left corner of the new location and click MB1 to complete the copy. The image in the original location is unchanged and a copy is placed in the new location. To cancel a copy operation once the original area has been selected, select the copy icon again.

Move

Similar to the copy operation. However, after moving the area to a new location, the original area is removed by filling it with the currently selected color.

Flood Fill

To flood fill an area, choose a location and click MB1. All contiguous pixels of the same color are changed to the current color.

Selecting and adding colors

The color bar allows you to select which color each of the previous graphics operations uses.

The color bar, located between the editable pixmap and the control panel, is a bar of rectangles, each of which corresponds to one color in the pixmap. The (+) button at the top of the bar allows you to add colors to the bar.

The current color is denoted by a check mark. When you select a new color it becomes the current color. To add a new color to the color bar, click on the (+) button at the top of the bar. To change a color, double-click on that color in the color bar. The Color Editor dialog, allowing you to change that color, is displayed.

Resizing the pixmap

To resize the pixmap, press MB1 on an edge of the small pixmap display, drag it to a new location, and release the mouse button. The size of the new image is displayed (in pixels) in the upper left corner of the small pixmap. When the image is made bigger all new pixels are set to the current color. When the image is made smaller only the upper left portion of the image that fits on the screen is saved.

Panning and zooming

The Pixmap Panner allows you to zoom and pan the pixmap. Grabbing the Pixmap Panner with MB1 allows you to move your current view of the pixmap to a new location. For example, dragging the Pixmap Panner to the extreme top right of its bounding area allows you to see the extreme top right corner of the pixmap. If the Pixmap Panner completely fills the area, then the entire pixmap is currently shown and the Pixmap Panner cannot be moved. When the Pixmap Panner has the keyboard focus, it can also be moved by using the arrow keys.

To zoom the image in and out, use the up and down arrow buttons. This changes the magnification of the drawing pixmap. A magnification of 8x means that each pixel in the real pixmap is represented by an 8x8 rectangle in the pixmap editor. Zooming in and out does not change the image in any way, it only affects your view of the image.

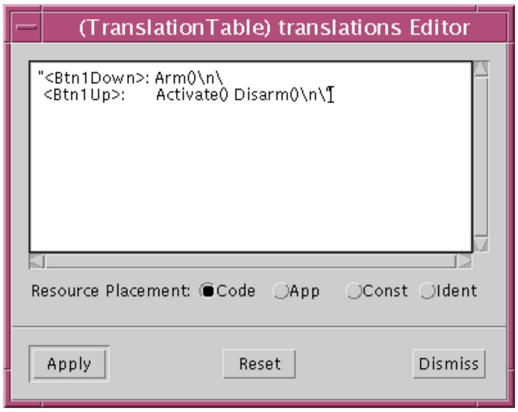
Pixmap Editor dialog

Two buttons appear on the bottom of the Pixmap Editor dialog:

- Apply sets the currently selected pixmap to the pixmap in the drawing area and removes the Pixmap Editor.
- Dismiss removes the Pixmap Editor without updating the pixmap.

Translation Table Editor

Allows you to construct tables of translations (mappings of user actions to widget functions) for the widgets in your interface:



```
"<Btn1Down>: Arm()\n\
 <Btn1Up>:     Activate() Disarm()\n\
```

Resource Placement: [✓ Code] App Const Ident

Figure 142. Translation Table Editor

Type your translation table into the editor. Do not include “\n” for line breaks; they are inserted automatically.

Note: Certain actions are provided by the widgets themselves. If you wish to use your own actions, you must first register an action table with the Translation Manager. For further information, refer to the X Toolkit documentation.

Widget Editor

Some resources, particularly many of the constraint resources, take a widget instance name as a value. The Widget Editor displays a list of valid widgets.

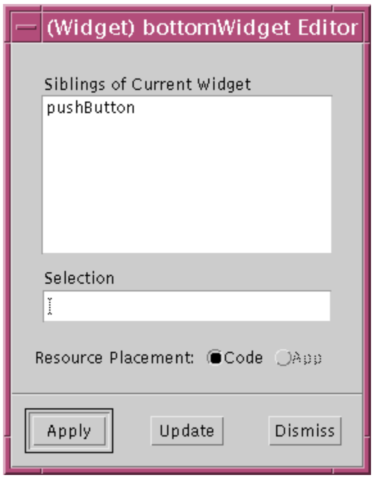

(Widget) bottomWidget Editor
Siblings of Current Widget
pushButton
Selection
Resource Placement: ● Code ○ App
Apply
Update
Dismiss

Figure 143. Widget Editor

Above the widget list is a label describing the relationship between the selected widget and the widgets in the list. To choose a widget, click on the widget name.

Resources

A

Overview

This appendix describes the Builder Xcessory resources and includes the following sections:

- **Builder Xcessory Resources** on pages 252 to 270
- **Builder Xcessory Application Resource Defaults** on page 271
- **Builder Xcessory Resource Command Line Options** on page 276
- **Resources Modifiable from Builder Xcessory** on page 277

Builder Xcessory Resources

actionCounts

Specifies the number of Button/Key actions to count between automatic saves of the interface. Setting this value to 0 disables the autosave feature. (Autosave Interval text field located on the General tab of the Browser User Preferences menu.)

alphabeticOrder

Specifies whether resources in the Resource Editor will be listed in alphabetical order. If True, the resources are displayed in alphabetical order sorted by their resource names. If False, the resources are displayed in the widget internal order. (Alphabetical Order toggle on the Options menu of the Resource Editor.)

alwaysGeneratePixmaps

Specifies whether pixmaps will always be output, even if none are referenced in the generated application. If True, all pixmaps are output always, whether referenced or not. If False, all pixmaps are generated only if at least one of them is referenced in the generated application. (Always Generate Pixmaps on the Save File tab of the Browser User Preferences menu.)

autoCheckOut

Specifies Builder Xcessory's response when you attempt to Open or Read a UIL file which has read-only file permissions, If True, you are prompted to check the file out from the version control system before reading it into Builder Xcessory. If False, Builder Xcessory reads the file and marks it as read-only.

autoDismissStartup

Automatically dismisses the Startup Panel when you start Builder Xcessory. Set on the User Preferences Behavior tab from the Browser Options menu.

autoMenuPopup

Specifies whether or not the associated menu will pop up when you select a Cascade Button, Option Menu, or Popup Menu. If True, the menu pops up automatically, as if the Show Menu command had been executed. If False, the menu does not pop up.

browserState

Specifies the geometry and iconification state of the Browser to use the next time Builder Xcessory runs. The string has the following format:

```
<width>x<height>+<x>+<y>:[normal|iconic]
```

brwsrToolbar

Specifies a list of items to place on the Browser tool bar. The list is comma separated and also specifies the order of placement. Unrecognized items are silently ignored. The last occurrence of a duplicate entry is used. Valid values include the following:

alignbtm	class	gilpref	open	saveclass
aligned	classhier	helpindex	palette	saveclassas
alignhctr	clearmsgs	i18npref	paste	selparent
alignlft	compchild	iconify	play	settings
alignrgt	constmgr	identmgr	print	showlists
aligntop	copy	importgil	printset	showtree
alignvctr	cut	keepparent	procmgr	snapgrid
appmgr	debug	lockclass	raise	stylemgr
apppref	deiconify	lower	read	subclass
brwssearch	delete	mainbrowser	readclass	toolpref
btoolbar	editfile	makeapp	receptor	topal
build	enlarge	menus	revert	typemgr
checkin	exit	messsages	rsceditor	unclass
checkout	filemgr	natural	save	unlockclass
chooselang	gencode	new	saveas	userpref

buttonOneShellString

Specifies the shell type to use when a widget is placed outside any parent using MB1. Valid options are ApplicationShell, TopLevelShell, TransientShell, and XmDialogShell.

buttonThreeShellString

Specifies the shell type to use when a widget is placed using MB3. Valid options are ApplicationShell, TopLevelShell, TransientShell, and XmDialogShell.

centerDialogs

Specifies whether the Builder Xcessory message dialogs should popup under the cursor or in the center of the screen.

clearcaseCCOCommand

The command to issue when the user requests a Cancel Check Out and configured the Source Code Control to ClearCase.

clearcaseCICommand

ClearCase Check In command.

clearcaseCOUCommand

ClearCase Check Out Unlocked command.

clearcaseCommand

ClearCase Check Out Locked command.

clearcaseHDRInfo

String to insert into all UIL files for ClearCase logging identification. Default is “ ”.

confirmWidgetDeletes

If True (default), displays a dialog requesting confirmation for a delete operation on the interface. If False, confirmation is not required to delete an object.

debug

Specifies whether the Builder Xcessory outputs additional debugging information.

debugger

Specifies the debugger and application build tool to use when you select Debug Mode or Build Application in the Browser. Has the following possible values:

- Environment
  User's development environment debugger
- CenterLine
  CenterLine debugging environment (CodeCenter/ObjectCenter)

defaultBxType

Specifies the default type when you create an identifier in the Identifier Manager.

defaultGilArgs

When importing a GIL file, specifies the arguments passed to the GIL translator.

defaultGilTranslator

When importing a GIL file, specifies the GIL code translator.

defaultLanguage

Specifies the language selected in the Language dialog of the Browser Options menu

defaultPlacement

Specifies the list of resource data types that default to App placement. A comma-separated list of resource data type names (all lowercase) is supplied to the data.

delayShellRealize

If False, all shells of an interface are displayed when it is loaded; the hierarchy of every shell is displayed in the Browser. If True, the shells and their hierarchies are not displayed. Instead, the shell names are listed on the Browser, and you must click on each shell name to display it. This is most useful when you are working with a complex interface and need to conserve screen real estate. When true, the interface will also load faster.

demo

When the Builder Xcessory starts, it searches for the license data file. If it cannot find the license data file, it will not allow the user to save any work. This option allows Builder Xcessory to forego the search and run in demo mode. In demo mode, UIL code generation, autosave, and the Save/Save As features are disabled.

directoryPath

Specifies the directory where files will be written. This is initially the directory from where Builder Xcessory is started, but will change as UIL files are opened. It can also be changed explicitly.

editor

Specifies which editor to use: Nedit, Vi, Emacs, or User Environment.

emacsCommand

The command to use when the editor is Emacs.

file

Specifies the file to load immediately after initialization. This is most useful when used with its command line variant.

firstNodeStateClosed

Specifies whether the Browser tree will start any hierarchy with the node closed. This is a useful setting when reading large hierarchies as it helps improve the Builder Xcessory startup time.

force

This resource is used only in conjunction with the load resource. Specifies that Builder Xcessory should generate Palette entries for any and all user defined widgets, even if they already appear in a WML file.

generateUnmodifiedDefaults

When Builder Xcessory is asked to place a resource value in the application defaults file, on rare occasions, it is unable to supply a valid string for the output value. If this resource is set to False, Builder Xcessory will skip over the resource specification. If it is set to True, Builder Xcessory will print a resource specification to the app-defaults file without a value string following the colon.

gilLookFeel

Sets Builder Xcessory to import GIL interface objects as having a Motif or OPENLOOK look and feel.

gilReposition

Specifies whether Builder Xcessory lays out the interface based on size changes encountered in Motif version of OPENLOOK objects.

gridUnit

Specifies the size, in pixels, of the grid to be used when snapToGrid is True.

hierarchyAutopanTimeout

Specifies the time in milliseconds before the Panner will automatically pan during a drag and drop operation on an object instance hierarchy in the Browser.

identifierLegalChars

The set of non-alphanumeric characters to accept as valid in identifier names. Default `"_$.()[]&-><:"`

installErrorHandlers

Specifies whether Builder Xcessory should install its catastrophic error handlers which try to save the interface in progress when a catastrophic error occurs. This is only useful if a core file is the desired end product. Specifying

False for this resource will turn off the error handlers and produce a core file on a catastrophic error. If False, the interface will not be saved on catastrophic error.

integratePurify

Specifies whether memory test targets should be added to the various makefiles and imakefiles.

integrateXrunner

Specifies whether Xrunner targets should be added to the Makefile.

lieToWindowManager

Specifies whether Builder Xcessory uses USPosition to set x and y values. This is technically incorrect, but is necessary for most window managers to place the windows correctly.

load

Specifies whether Builder Xcessory should try to create a WML file and Palette entries for any user defined widgets.

loadICS

Specifies whether or not to load the EnhancementPak widgets onto the Palette during startup.

Note: The EnhancementPak widget set is a separate product. Please consult your ICS sales representative for further information.

localDirectory

Specifies the directory that Builder Xcessory uses for individual additions to the Palette, such as user collections. This directory must be writable.

matchPaletteColors

If True, Palette and toolbar pixmaps use the symbolic background and foreground colors. If False, the Palette and toolbar pixmaps are displayed with individual background colors.

memoryTool

Specifies which memory tool options to add to the application makefile: PURIFY, SENTINEL, or USER.

minimumWidgetHeight

Specifies the minimum height a widget can have when created.

minimumWidgetWidth

Specifies the minimum width a widget can have when created.

motifVersion

Specifies the version of the Motif toolkit for which Builder Xcessory will generate code. The Builder Xcessory will read in code written in any version of UIL.

neditCommand

Specifies the format of the command to use for the nedit editor.

newMrmStyle

Specifies how MRMRegisterArg is output and initialized in code when outputting UIL code. If True, MRMRegisterArg is output as an uninitialized array and is initialized in code. If False, MRMRegisterArg is output and initialized as a static array.

objectcenterCommandLine

Specifies the command line to start ObjectCenter.

paletteState

Specifies the geometry, iconification, view mode and catalog file of each Palette to display when Builder Xcessory is next executed. The string has the following format:

```
<width>x<height>+<x>+<y>:[normal|iconic]:<catalog
>:[outline|tab]:[IconOnly|IconNone|IconTop],...
```

pixmapBackground

Specifies the color to use in the Palette and toolbar pixmaps for the symbolic background color.

pixmapForeground

Specifies the color to use in the Palette and toolbar pixmaps for the symbolic foreground color.

pixmapMode

Specifies which set of pixmaps, if any, are used on the Palette. May be either BEST_FIT, FORCE_BW, or FORCE_COLOR. When pixmapMode is set to BEST_FIT, Builder Xcessory uses either the black and white pixmaps or the color pixmaps depending on the display. When pixmapMode is set to FORCE_BW, Builder Xcessory uses the black and white pixmaps. When pixmapMode is set to FORCE_COLOR, Builder Xcessory uses the color pixmaps.

printAppendText

When an object instance hierarchy is printed to the screen, this resource determines how the text will be displayed in the output window. When True, the text from the hierarchy will be appended to the output window. If False, the text will replace the current text in the output window.

printClassNames

When an object instance hierarchy is printed. this resource determines whether or not the class names of widgets will be printed. When True, the class names will be printed. If False, they will not.

printCommand

This resource determines what command is used to print an object instance hierarchy to your printer. By default, the print command is lp.

printExpandUserClasses

When an object instance hierarchy is printed to the screen. this resource determines how much information about class instances is printed. If True, information about all of the widgets that make up any user-defined class is printed. If False, only information about the class instance is printed

printIndentAmount

When an object instance hierarchy is printed to the screen. this resource determines how many spaces to indent for each child level. The default value is 4.

printInstanceNames

When an object instance hierarchy is printed. this resource determines whether or not the instance names of the objects in the hierarchy should be printed. When True, the instance names are printed. If False, they are not.

printSelected

When an object instance hierarchy is printed. this resource determines how many of the objects in the Browser window are printed. When True, only the selected object(s) and their descendents are printed. If False, all objects that are currently displayed on the Browser are printed.

printToPrinter

When an object instance hierarchy is printed, determines the result of selecting Print Hierarchy from the File menu on the Browser. If True, the hierarchy is printed by your printer. If False, it is not.

printToWidget

Determines the result of selecting Print Hierarchy from the File menu on the Browser. If True, the object instance hierarchy is printed to an output window. If False, it is not.

printXtHierarchy

When an object instance hierarchy is printed. this resource determines the format in which the hierarchy is printed. When False, the hierarchy is printed as interpreted by Builder Xcessory. If True, the hierarchy is printed as it is implemented in Xt.

procLegalChars

The set of non-alphanumeric characters to accept as valid in procedure names. Default: `“_$<>:”`

purifyCommand

Specifies the command used to make a memory test target in the application makefile.

raiseRscShellOnDoubleClick

Specifies whether the Resource Editor should be raised to the top of the stacking order whenever a widget is updated.

rcsCCOCommand

Specifies the command line for the RCS Cancel Check Out command.

rcsCICommand

Specifies the command line for the RCS Check In command.

rcsCommand

Specifies the command line for the RCS Check Out Locked command.

rcsCOUCommand

Specifies command line for RCS Check Out Unlocked command.

rcsHDRInfo

String to insert into all UIL files for RCS source code control login identification. Default is “$ID$”

removeNewline

Specifies whether the Resource Editor will remove the last newline at the end of a string, typically typed in accidentally.

resourceEditorAutoUpdate

Specifies whether the Resource Editor will be in Automatic Update mode or not. If True, the Resource Editor will update its display automatically each time a widget instance is selected. If False, the editor will only be updated on request. The default value is False.

resourceEditorView

Specifies the Resource Editor view. This setting determines which group of resources is displayed—all resources are displayed by default. Other settings include simple, programmer, modified, and not equal (for use with multiply-selected instances). You usually select these from the Resource Editor Resources menu.

reToolbar

This resource is a list of the items to place on the Resource Editor tool bar. The list is comma separated and also specifies the order of placement. Unrecognized items are silently ignored. The last occurrence of a duplicate entry is used. Valid values include the following:

allform	dfltplace	modform	search	update
alphabetic	gad	neqform	simpleform	wid
autoupdate	gen_class	progform	storage	
classfile	header	ref_class	typed	
creation	manage	retoolbar	unmanage	

rscEditorState

Specifies the geometry and iconification state of the Resource Editor the next time Builder Xcessory executes. The string has the following format:

```
<width>x<height>+<x>+<y>:[normal|iconic]
```

saberCMessage

Specifies the message Builder Xcessory sends to CodeCenter when CenterLine Mode is invoked.

saberCPlusMessage

Specifies the message Builder Xcessory sends to ObjectCenter when Centerline Mode is invoked.

saberCommandLine

Specifies the command line Builder Xcessory uses to invoke CodeCenter.

saberExec

Specifies whether Builder Xcessory should execute CodeCenter when CenterLine mode is invoked.

saberResetMessage

Specifies the message Builder Xcessory sends to CodeCenter to reset CenterLine Mode.

saberUILMessage

Specifies the message Builder Xcessory sends to ObjectCenter when Centerline Mode is invoked.

saveState

Specifies whether Builder Xcessory should save Browser, Resource Editor, and Palette states on exit. Valid values are SaveState or UseDefault.

sccCCOCommand

Specifies the command line for the User Specified Cancel Check Out command.

sccCICommand

Specifies the command line for the User Specified Check In command.

sccCommand

Specifies the command line for the User Specified Check Out Locked command.

sccCOUCommand

Specifies the command line for the User Specified Check Out Unlocked command.

sccHDRInfo

String to insert into all UIL files for user-defined source code control logging identification. Default is “ ”.

sccsCCOCommand

Specifies the command line for the SCCS Cancel Check Out command.

sccsCICommand

Specifies the command line for the SCCS Check In command.

sccsCommand

Specifies the command line for the SCCS Check Out Locked command.

sccsCOUCommand

Specifies the command line for the SCCS Check Out Unlocked command.

sccsHDRInfo

String to insert into all UIL files for RCS logging identification. Default: “%W% %D% %T%”

sentinelCommand

Memory test command to use when the memory tool is set to SENTINEL.

setPortholeLocation

When True, this resource will cause Builder Xcessory to center the Browser tree automatically about the selected object node. When False, the tree will not be moved when an object is selected.

showBrowserToolbar

If True, specifies that the Browser Toolbar should be displayed at start up time. If False, the Browser Toolbar will be hidden initially.

showCompositeChildren

When True, the compound children of any compound widgets are displayed in the Browser tree hierarchy. If False, compound children will not be displayed.

showInstanceData

If True, specifies that the Resource Editor Instance Data Field header should be shown at start up. If False, the Resource Editor Instance Data Field Header is hidden initially.

showMessages

When True, this resource will cause Builder Xcessory to start with the message window in the Browser displayed rather than hidden.

showRscEdSearch

If True, specifies that the Resource Editor Search Field should be shown at start up. If False, the Resource Editor Search Field is hidden initially.

showRscEdToolbar

If True, specifies that the Resource Editor Instance Data Field header should be shown at start up. If False, the Resource Editor Instance Data Field Header is initially hidden.

showWidgetSearch

If True, specifies that the Browser Widget Search/Select Field be shown at start up. If False, the Browser Widget Search/Select Field is initially hidden.

snapToGrid

Specifies whether widgets are placed in accordance with the grid. See also gridUnit.

sourceCodeControl

Specifies which Source Code Control (or version control) system to use for Check In and Check Out requests.

Possible values include the following:

CVS	Use CVS source code control commands.
ENV	Use the user's development environment source code control tool.
RCS	Use RCS source code control commands.
SCCS	Use SCCS source code control commands.
USER	Use User-specified source code control commands.

strictDialogs

Specifies whether Builder Xcessory should allow only widgets that are subclasses of XmBulletinBoard to be the child of an XmDialogShell.

symbolicUIL

Specifies whether Builder Xcessory should use symbolic names when generating UIL code. If True, and motifVersion is 1.0, the output UIL file will include the file XmAppl.uil. If False, this file will not be included and Builder Xcessory will generate the actual constant values instead of the symbolic names.

system

When load is True, specifies that the WML file and Palette entries should be put in the system directory instead of the local directory. Typically, root privileges are necessary for this option.

systemDirectory

Specifies the directory that Builder Xcessory searches for its system initialization files.

toolEnvironment

Specifies that the Builder Xcessory integration with a development environment should be used. Valid values include:

None	do not communicate with an IDE.
SUNTT	communicate with SunSoft WorkShop (version 2 or 3)
SGITT	communicate with SGI Developer Magic

Note: Communication with the DEC FUSE environment will only work when Builder Xcessory is started from the DEC FUSE Control Panel.

toolbarButtonMode

Specifies how the buttons in both the Browser and Resource Editor Toolbars should be displayed.

Possible values include the following:

- XiIconLeft—Display both labels and icons
- XiIconNone—Display labels only
- XiIconOnly—Display icons only

typeLegalChars

The set of non-alphanumeric characters to accept as valid in type names. Default values are “_ $ < > &”

typeOrder

Specifies whether resources in the Resource Editor will be listed in order of their type. If True, the resources are ordered according to their resource type. If alphabeticOrder is also True, then the resources are ordered alphabetically within each resource of the same type.

uilIncludePath

Specifies the directory or directories that Builder Xcessory searches for files included by the uil include directive. Additional include directories may be specified by separating with a comma.

uilPixmapBehavior

This resource affects the generation of icon values in UIL code. If True, the icon’s color table will be generated in a separate value statement immediately preceding the icon value to which it applies. If False, the icon’s color table will be generated as a part of the icon value. This is a legal UIL value specification, but most UIL compilers flag this syntax as an error or do not compile it correctly.

uniqueSubclassNames

If True, prepends the subclass name and an underscore (_) to all subclass widget names in order to make them unique from their corresponding superclass widgets. If False, allows subclass widgets to have the same names as their corresponding superclass widgets—you must then rename the subclass widgets yourself.

useOpenGL

Note: Valid on SGI Platforms only.

If True, load the OpenGL Drawing Area widget on the Palette. If False (default), load the IRIS GL Drawing Area widget on the Palette.

userDefinedUIL

Useful for users generating UIL code and unable to extend the UIL compiler to understand new widgets. Set to True only when generating the UIL file that will be compiled because the resulting UIL file becomes much larger and more confusing.

If True, writes UIL files with all non-motif widgets fully specified as user-defined UIL objects.

If False (default), writes UIL files treating all widgets as if defined to the UIL compiler.

userEditorCommand

Specifies the command to use for a user-defined editor.

userMemCommand

Specifies the command to use for a user-specified memory test tool.

viCommand

Specifies the format of the command to use for the vi editor.

viewSingleClass

If True, shows only one class at a time in the Browser and Class Hierarchy, when in Classes view. If False, show any number of classes at the same time in the Browser and Class Hierarchy.

uilWrapLines

Specifies whether or not long strings should be broken into shorter lines during UIL code generation. Some compilers cannot handle lines longer than 132 or 256 characters, a limit often exceeded by font lists or multi-font label strings. This option lets the user break these into multiple lines separated with a backslash and return character sequence.

wmPositionIsFrame

Specifies whether the window manager interprets X, Y locations relative to the window decorations. By default, most window managers take a window’s X and Y location as relative to the decorations. Change this value if you notice that the main Builder Xcessory windows are not restoring their positions correctly from session to session.

xrunnerCommand

Specifies the libraries used to make an Xrunner object.

xtInstanceNameCompliant

If False, allows instance names of widgets to begin with an upper case letter. If True, forces a lower case letter. Capitalized widget names may cause problems with the proper inheritance of application defaults resource values and conflicts with Xt specifications.

Builder Xcessory Application Resource Defaults

The following table alphabetically lists the default values for Builder Xcessory application resources:

Resource	**Default**
actionCounts	100
alphabeticOrder	True
alwaysGeneratePixmaps	True
autoDismissStartup	False
autoMenuPopup	True
browserState	“:normal”
brwsrToolbar	“open, save, cut, copy, paste, delete”
buttonOneShellString	“TopLevelShell”
buttonThreeShellString	“XmDialogShell”
clearcaseCCOCommand	**Note:** You can access these statements from the Source Code Control tab of the Browser Tools Customization dialog.(Select Use ClearCase) (These are very long, complicated statements that do not fit in this table.)
clearcaseCICommand	
clearcaseCOUCommand	
clearcaseCommand	
cvsCIComand	cvs commit %version [-r%s] %comments [-m”%s”] %file[%s]
debug	False
debugger	“environment”
defaultBxType	“Undeclared”
defaultLanguage	““
defaultGilTranslator	`giltouil`
defaultPlacement	“”
delayShellRealize	False
demo	False
directoryPath	Dynamic[a]
editor	“Environment”
emacsCommand	`“emacsclient %line[+%s] %file[%s]”`
file	Dynamic[b]
firstNodeStateClosed	True

Resource (Continued)	**Default**
force	False
generateUnmodifiedDefaults	False
gilLookFeel	Motif
gilReposition	Yes
gridUnit	10
hierarchyAutopanTimeout	1000
identifierLegalChars‘	`“_$.()[]&-><:”`
installErrorHandlers	True
integratePurify	False
integrateXrunner	False
lieToWindowManager	True
load	False
loadDEC	True
loadICS	False
localDirectory	`${HOME}/.builderXcessory`
matchPaletteColors	False
memoryTool	“PURIFY”
minimumWidgetHeight	20
minimumWidgetWidth	20
motifVersion	2.1
newMrmStyle	True
objectcenterCommandLine	`objectcenter -motif`
pixmapBackground	grey 80
pixmapForeground	black
pixmapMode	BEST_FIT
printAppendText	False
printClassNames	True
printCommand	“lp”
printExpandUserClasses	False
printIndentAmount	4
printInstanceNames	True
printSelected	True

Resource (Continued)	Default
printToPrinter	False
printToWidget	True
printXtHierarchy	`False`
procLegalChars	`"_$<>:"`
purifyCommand	`purify`
raiseRscShellOnDoubleClick	False
rcsCCOCommand	`ci -q %overwrite[-f] %keep[-u]] %version[-r%s]`
rcsCICommand	`ci -q %overwrite[-f]%keep[-u] %version[-r%s]`
rcsCommand	`co -q -l %overwrite[-f] %version[-r%s]%file[%s]`
rscCOUCommand	`co -q %overwrite[-f] %version[-r%s] %file[%s]`
rscHDRInfo	`"$Id$"`
removeNewline	True
resourceEditorAutoUpdate	False
resourceEditorView	simple
reToolbar	"update"
rscEditorState	":normal"
saberCMessage	`directory | make -f %makefile codecenter | run`
saberCPlusMessage	`directory| make -f %makefile objectcenter | run`
saberCommandLine	`codecenter -motif`
saberExec	False
saberResetMessage	`reinit | reset`
sccCCOCommand	““
sccCICommand	““
sccCommand	““
sccCOUCommand	““
sccHDRInfo	““
sccsCCOCommand	`sccs unedit %file[%s]`

Resource (Continued)	Default
sccsCIcommand	`sccs delta -s %comments[-y\"%s\"] %file[%s] %keep[;sccs get -s %file[%s]]`
sccsCommand	`sccs get -e -s %version[-r%s] %file[%s]`
sccsCOUCommand	`sccs get -s %version[-r%s] %file[%s]`
sccsHDRInfo	`"%W% %D% %T%"`
sentinelCommand	"memadvise"
setPortholeLocation	False
showBrowserToolbar	True
showCompositeChildren	False
showInstanceData	True
showMessages	False
showRscEdSearch	True
showRscEdToolbar	True
showWidgetSearch	False
snapToGrid	True
sourceCodeControl	"ENV"
strictDialogs	True
symbolicUIL	True
system	False
systemDirectory	`/usr/lib/X11/builderXcessory`
toolEnvironment	"None"
toolbarButtonMode	XiIconLeft
typeLegalChars	`"_$<>&"`
typeOrder	False
uilIncludePath	`/usr/lib/X11/builderXcessory`
uilPixmapBehavior	True
uilWrapLines	True
uniqueSubclassNames	True
useOpenGL	False
userDefinedUIL	False

Resource (Continued)	Default
userEditorCommand	““
userMemCommand	““
viCommand	`xterm -e vi %line[+%s] %file[%s]`
viewSingleClass	False
wmPositionIsFrame	False (on IRIX) True (elsewhere)
xrunnerCommand	`“$(M_ROOT)/arch/X11.5/Xm1.2/mic_xm.o”`
xtInstanceNameCompliant	True

a. Value set to the directory where the project file is read from or written to.
b. Ibid.

Builder Xcessory Resource Command Line Options

The following table alphabetically lists the available command line options for Builder Xcessory application resources:

Builder Xcessory Resource	Command Line Option
debug	-debug
demo	-demo
file	-file <filename>
force	-force
installErrorHandlers	-core
largeIcons	-largeIcons
load	-load
localDirectory	-local_directory <path>
system	-system
systemDirectory	-system_directory <path>
toolEnvironment	-fuse -devmagic -workshop
useOpenGL	-openGL/+openGL
(no resource)	-version
wmPositionIsFrame	-positionFrame/+positionFrame

Table Key

The following table defines the key for *"Resources Modifiable from Builder Xcessory"* on page 277:

Symbol	Description
B	Browser Menu Bar
DRT	Default Resource Type Dialog
CGP	Code Preferences Dialog
PR	Print Setup Dialog
RE	Resource Editor Menu Bar
TP	Tools Preferences Dialog
UP	User Preferences Dialog

Resources Modifiable from Builder Xcessory

The following table alphabetically lists the resources that are modifiable from Builder Xcessory:

Resource	Location	Saved To
actionCounts	(UP) Autosave Interval	`.bxrc`
alphabeticOrder	(RE) Options menu	`.bxrc`
alwaysGeneratePixmaps	(UP) Generate Pixmaps	`.bxrc`
autoCheckOut	(TP) Check Out Read-Only Files	`.bxrc`
autoDismissStartup	(UP) Auto Dismiss Startup Panel	`.bxrc`
autoMenuPopup	(UP) Auto Menu Popup	`.bxrc`
brwsrToolbar	(B) View Menu	`.bxrc`
buttonOneShellString	(UP) Button One Shell	`.bxrc`
buttonThreeShellString	(UP) Button Three Shell	`.bxrc`
debugger	(TP) Debugger & Build Manager	`.bxrc`
defaultPlacement	(DRT)	`.bxrc`
delayShellRealize	(UP) Delay Shell Realize	`.bxrc`
directoryPath	(CGP) Directory Path	N/A
emacsCommand	(TP) Use Emacs	`.bxrc`
file	(B) Open	N/A
firstNodeStateClosed	(UP) Tree Initially Closed	`.bxrc`
gilLookFeel	(GD)	`.bxrc`
gilReposition	(GD)	`.bxrc`
gridUnit	(UP) Grid Units	`.bxrc`
hierarchyAutopanTimeout	(UP) Panner Timeout	`.bxrc`
includeInfo	(CGP) C Output Include Information	UIL File
integratePurify	(TP) Purify Command	`.bxrc`
integrateXrunner	(TP) XRunner Library	`.bxrc`
motifVersion	(UP) Motif Version	`.bxrc`
objectcenterCommandLine	(TP) C++ Command	`.bxrc`
printAppendText	(PR)	`.bxrc`
printClassNames	(PR)	`.bxrc`
printCommand	(PR)	`.bxrc`

Resource (Continued)	Location	Saved To
printExpandUserClasses	(PR)	`.bxrc`
printIndentAmount	(PR)	`.bxrc`
printInstanceNames	(PR)	`.bxrc`
printSelected	(PR)	`.bxrc`
printToPrinter	(PR)	`.bxrc`
printToWidget	(PR)	`.bxrc`
printXtHierarchy	(PR)	`.bxrc`
purifyCommand	(TP) Purify Command	`.bxrc`
raiseRscShellOnDoubleClick	(UP) Raise Shell	`.bxrc`
rcsCCOCommand	(TP) Cancel Check Out Command	`.bxrc`
rcsCICommand	(TP) Check In Command	`.bxrc`
rcsCommand	(TP) Check Out Locked Command	`.bxrc`
rcsCOUCommand	(TP) Check Out Unlocked Command	`.bxrc`
rcsHDRInfo	(TP)	`.bxrc`
reToolbar	(RE)	`.bxrc`
saberCMessage	(TP) C Message	`.bxrc`
saberCPlusMessage	(TP) C++ Message	`.bxrc`
saberCommandLine	(TP) C Command	`.bxrc`
saberExec	(TP) Execute CenterLine Command	`.bxrc`
saberResetMessage	(TP) Reset Message	`.bxrc`
sccCCOCommand	(TP) Cancel Check Out Command	`.bxrc`
sccCICommand	(TP) Check In Command	`.bxrc`
sccCommand	(TP) Check Out Locked Command	`.bxrc`
sccCOUCommand	(TP) Check Out Unlocked Command	`.bxrc`
sccsCCOCommand	(TP) Cancel Check Out Command	`.bxrc`
sccsCICommand	(TP) Check In Command	`.bxrc`
sccsCommand	(TP) Check Out Locked Command	`.bxrc`
sccsCOUCommand	(TP) Check Out Unlocked Command	`.bxrc`
sccsHDRInfo	(TP)	`.bxrc`
sentinelCommand	(TP)	`.bxrc`
setPortholeLocation	(UP) Center Browser Tree	`.bxrc`
showBrowserToolbar	(B) View Menu... Show Toolbar	`.bxrc`

Resource (Continued)	Location	Saved To
showCompositeChildren	(B) View Menu... Show Compound Children	`.bxrc`
showInstanceData	(RE) View Menu... Show Header Data	`.bxrc`
showMessages	(B) View Menu... Show Messages	N/A
showRscEdSearch	(RE) View Menu... Show Search Bar	`.bxrc`
showRscEdToolbar	(RE) View Menu... Show Tool Bar	`.bxrc`
showWidgetSearch	(B) View Menu... Show Search/ Select	`.bxrc`
snapToGrid	(B) View Menu... Snap to Grid	`.bxrc`
sourceCodeControl	(TP) Source Code Control	`.bxrc`
strictDialogs	(UP) Strict Dialogs	`.bxrc`
toolbarButtonMode	(UP) Tool Bar Display Mode	`.bxrc`
typeOrder	(RE) View Menu... Type Order	`.bxrc`
uil	(MC) UIL	`.bxrc`
userDefinedUIL	(UP) Special UIL	`.bxrc`
userEditorCommand	(TP)	`.bxrc`
userMenuCommand	(TP)	`.bxrc`
viCommand	(TP)	`.bxrc`
viewSingleCLass	(UP)	`.bxrc`
xrunnerCommand	(TP) XRunner Library	`.bxrc`
xtInstanceNameCompliant	(UP)	`.bxrc`

Palette Objects

B

Overview

This appendix describes each Palette object in the following sections:

- **Motif Xm Widgets** on page 283
- **ViewKit ObjectPak Vk Classes** on page 300
- **EnhancementPak Xi Widgets** on page 314
- **Java Awt Objects** on page 321
- **SGI Sgm Widgets** on page 329
- **CDE Dt Widgets** on page 336

Palette object icons table

Depending on the currently selected language and the platform on which you are running Builder Xcessory, the Palette displays labeled, iconic representations of Motif widgets, ViewKit objects, a subset of the EnhancementPak widget set, Java objects, and platform-specific objects. The following table lists the objects displayed for specific platforms and languages:

Objects	Platforms	Languages
Motif Xm widgets	All	C++, C, ViewKit, and UIL
EnhancementPak Xi widgets	All	C++, C, ViewKit, and UIL
ViewKit ObjectPak Vk objects	All	ViewKit only
Java Awt objects	All	Java only
SGI Sgm widgets	SGI	C++, C, ViewKit, and UIL
CDE Dt widgets	AIX, Solaris, HP-UX 10+, and DEC UNIX 4	C++, C, ViewKit, and UIL

Note: If you are using BX PRO, you can use and compile the EnhancementPak and ViewKit ObjectPak objects in your interface. If you are using Builder Xcessory, you can use the EnhancementPak and ViewKit ObjectPak objects in your interface, but you must purchase the respective libraries to compile any interface built with the EnhancementPak and ViewKit objects. Contact your ICS Sales Representative for more information.

Motif Xm Widgets

Motif widgets are displayed on the Palette for all languages except Java and for all platforms. Initially, Motif widgets are grouped in the following folders:

- Primitives

  XmToggleButton, XmPushButton, XmDrawnButton, XmArrowButton, XmLabel, XmSeparator, XmScrollBar, XmTextField, XmText, XmScrolled-Text, XmList, and XmScrolledList.

- Containers

  XmMainWindow, XmScrolledWindow, XmPanedWindow, XmScale, XmForm, XmBulletinBoard, XmDrawingArea, XmRowColumn, XmRadioBox, and XmFrame.

- Menus

  XmMenuBar, XmPulldownMenu, XmOptionMenu, and XmPopupMenu.

- Dialogs

  XmSelectionBox, XmFileSelector, XmMessageBox, and XmCommand.

The following sections provide descriptions of each Motif widget in alphabetical order (refer to your Motif documentation for more detailed information).

Arrow Button

Description

The XmArrowButton consists of a directional arrow surrounded by a border shadow. When the widget is selected, the shadow changes and the Arrow Button appears to be depressed. When the Arrow Button is unselected, the shadow changes and the Arrow Button appears to be released. Callbacks notify the application when an Arrow Button is activated.

Notes

Use for application actions such as moving up in a text editor or searching forward through the text buffer.

Bulletin Board

Description

The XmBulletinBoard widget is a composite widget that provides simple geometry management for children. It does not force positioning on its children, but can be set to reject geometry requests that result in overlapping children. Bulletin Board is used as a general container widget.

Notes

- Use when your interface calls for exact positioning by specific location of the children objects.
- Positions children at the requested locations
- Can be set to move children out of its margins or so that they do not overlap one another
- Basis for Motif dialogs

ComboBox

Description

The XmComboBox widget provides the capabilities of a text field and a drop down list. Provides funtionality similar to the Option Menu.

Notes

Displays a type-in area showing the current value and a popup list containing other possibilities.

Command

Description

The XmCommand widget is a special-purpose composite widget for command entry that provides a built-in command-history mechanism. Provides the user with a method for entering commands, while preserving a command history.

Notes

Includes the following:

- Command-line text-input field
- Command-line prompt
- Command-history list region
- Several auxiliary functions for better usability

Container

Description

The XmContainer widget manages child widgets that have the container item trait. It offers several different layout forms for its children.

Notes

- Allows children to be displayed as a connected outline
- Controls size of children icons
- Shows a detailed view of children

Drawing Area

Description

The XmDrawingArea widget is an empty container easily adaptable to a variety of purposes. Does no drawing and defines no behavior except for invoking callbacks. Callbacks notify the application when graphics must be drawn through exposure and widget resize events and when the widget receives input from the keyboard or mouse.

Notes

- Use when your interface calls for an area containing graphics that you control.
- May also accept children.
- Uses geometry management rules similar to that of the Bulletin Board.

Drawn Button

Description

The XmDrawnButton widget consists of an empty window surrounded by a shadow border. It provides the application developer with a graphics area that can have Push Button input semantics. Callbacks notify the application when the widget is resized or receives an exposure event.

Notes

- Your application will be notified when it should draw particular graphics into the Drawn Button’s window.
- Callbacks notify the application of exposure, resize, and so forth.

FileSelector

Description

The XmFileSelector widget allows the user to select files from a hierarchy of directories and to traverse directories, viewing the files in these directories, and selecting files. The widget contains: a directory mask that includes a filter label and input field used to specify the directory; a scrollable list of directories; a scrollable list of filenames; a text input field; and push buttons for control.

Notes

Displays the following;

- Scrolled list of filenames
- Scrolled list of subdirectories
- Text input field for regular-expression matching
- Modifiable labels
- Several buttons for indicating the selection

Form

Description

The XmForm widget is a container that exerts strong control over its children's geometry. Constraints are placed on children of the Form to define attachments for each of the child's four sides. These attachments determine the layout behavior of the children when the Form resizes.

Notes

- Use for complex layouts in which children need to be placed with respect to one another.
- You can change the Form resources that specify the constraints on the children and how the children should be resized when the size of the Form changes.
- Nesting Forms within forms can help manage setting constraints.

Frame

Description

The XmFrame widget is a simple manager used to enclose a single work area in a border. The Frame widget uses the Manager class resources for border drawing and performs geometry management. The Frame widget's size always matches its child's size plus the margins defined for it.Frame may also control a second "title" child that can be used for labeling the contents of the work area.

Notes

- Use to display an etched edge around a child, either with or without an attached label.
- Use this object to visually separate parts of the application's interface.

IconGadget

Description

The XmIconGadget displays both simple text and a pixmap in several possible configurations.

Notes

Use when you want an object that has both the semantics of a Push Button or Toggle Button and the capability of displaying both simple text and a pixmap.

Label

Description

The XmLabel widget displays informational text or a pixmap image (which may optionally be surrounded by a margin).

Displays either text or pixmap and does not accept any button or keyboard input. Label widget receives enter and leave events and has a help callback.

Notes

- Text may be multi-line and multi-font.
- Does not respond to button or key input, and the text is not selectable.

List

Description

The XmList widget allows the user to choose one or more items from a group of text choices. Items are selected from the list using both the mouse and the keyboard. The currently selected items are highlighted. Clicking on the selected item activates a callback which receives information about the selected item. Several convenience routines are available.

Notes

Supports several interface styles that allow single or multiple selections of the items.

Main Window

Description

The XmMainWindow widget creates a main screen for an application that follows the Motif style guidelines.

Provides a Motif Style Guide compliant layout for the primary window of an application. This layout includes a Menu Bar, a Command window, a work region, a message window, and Scroll Bar widgets.

Notes

Manages the layout of a MenuBar, a command-entry area, a message area, a work region showing the main portion of the application, and scrollbars for displaying offscreen portions of the work region.

Menu Bar

Description

The XmMenuBar widget provides a permanent location for cascade buttons associated with the menus that appear either underneath the cursor or when the keyboard is used to invoke the menu system.

Notes

- Use in conjunction with several Pulldown Menus to create a Motif pulldown menu system.
- Menu Bar is a RowColumn widget with a specific resources set.

Message Box

Description

The XmMessageBox widget is a dialog class used for creating simple message dialogs. Convenience dialogs based on Message Box are provided for several common interaction tasks, which include giving information, asking questions, and reporting errors. A pixmap may be placed to the left of the message text to provide quick visual recognition of a familiar message.

The Template Dialog variant allows you to completely control the contents of the dialog (buttons, etc.) while maintaining compliance with the OSF/Motif Style Guide requirements on dialogs.

Notes

- Use for such tasks as reporting errors, supplying information, and asking simple questions.
- Contains the following a message symbol, message, and several buttons for indicating the response.
- Use the “dialogType” resource to change the display to the following:
  Message Dialog (default)
  Working Dialog
  Question Dialog
  Information Dialog
  Error Dialog
  Template Dialog

NoteBook

Description

The XmNoteBook widget manages a group of widgets such that only one widget in the group is visible at a time. Each child is associated with a page that displays text and/or a pixmap. Each page can have a tab assosiated with it. The user selects the tab, interactively determining which child is displayed. Tabs can be configured to appear above, below, and right or left of the work area, with the text oriented in any of the four cardinal directions. Visual resources can be changed to alter the apperance of the widget.

Notes

- Allows user to select tabs, either by pointer or by keyboard traversal.
- When a tab is selected, the tab appears to be raised above the other tabs and the child associated with the tab is made visible.
- One tab is selected at all times.
- The bindingPixmap can be set to NONE
- The backPageSize can be set to ZERO

Option Menu

Description

The XmOptionMenu widget provides the user with a choice of options from a list that is displayed.

Notes

- Displays the currently selected choice.
- Option Menu is a RowColumn widget with a specific resources set.

Paned Window

Description

The XmPanedWindow widget allows the user to vertically resize various portions of the application interface.

Composite widget that lays out children in a vertically tiled format. The first child is inserted at the top of the Paned Window and each successive child is inserted below. The Paned Window grows to match the width of its widest child and all other children are forced to this width. The height of the Paned Window is equal to the sum of the heights of all its children, the spacing between them, and the size of the top and bottom margins.

Notes

Each child object that the Paned Window controls is associated with a sash which the user may move to change the size of the child.

Popup Menu

Description

The XmPopupMenu widget creates a menu that provides the user with a set of choices or actions.

Notes

- Creates a menu that pops up dynamically.
- Popup Menu is a RowColumn widget with a specific resources set.

Pulldown Menu

Description

The XmPulldownMenu widget creates a pulldown menu system.

Notes

- Use one or more Pulldown Menus as the children of a MenuBar.
- The children of the Pulldown Menus may be simple buttons, cascade menus, or label objects that display choices or actions for the user.
- Must be a child of Menu Bar.
- Builder Xcessory automatically creates a Pulldown Menu and cascade button.

Push Button

Description

The XmPushButton widget issues commands within an application.

Consists of a text label or pixmap surrounded by a border shadow. When Push Button is selected, the widget appears as if it has been pressed in. When Push Button is unselected, it appears released.

Notes

- The visual capabilities are like those of the XmLabel.
- Typically displays a 3-D shadow, which gives it the raised appearance of a selectable object and which changes when selected to give the impression of being pressed in.

Radio Box

Description

Use the XmRadioBox widget with several Toggle Button children to offer the user options presented by the Toggle Buttons.

Notes

- Allows the user to select from several options.
- Radio Box is a RowColumn widget with a specific resources set.

Row Column

Description

The XmRowColumn widget is a general purpose manager capable of containing any widget type as a child. Generally does not require special knowledge of how its children function, and provides only support for several different layout styles. Can be configured as a Radio Box or one of four menu types, in which case, it expects only certain children. The four menu types are Menu Bar, Pulldown or Popup Menu Panes, and Option Menu.

Notes

- Supports several resources which specify the packing of the children.
- The Row Column is the basis for Motif menus and the Radio Box.

Scale

Description

The XmScale widget allows the user to select a value within a range.

Notes

- Used by an application to indicate a value from within a range of values.
- Scale is a manager widget, and can accept children.
- Children are evenly spaced across the top of the scale.

Scroll Bar

Description

The XmScrollBar widget allows the user to select a value within a range, such as a particular screen of text within a large file.

Allows the user to view an area that is too large to be displayed all at once. Scroll Bar widgets may be placed beside or within the widget that contains the data to be viewed. When the user interacts with the Scroll Bar, the area within the other widget scrolls.

Notes

- Displays two arrows for moving by a fixed increment.
- Displays a slider for moving by arbitrary amounts.

Scrolled List

Description

Use the XmScrolledList widget when the amount of data you need to display is larger than the area of the application available.

Notes

The object is an XmList within an XmScrolledWindow. That is, when you select Scrolled List from the Palette, two widget instances are created: List (the widget you selected) and Window (the parent of Scrolled List).

Scrolled Text

Description

Use the XmScrolledText widget when the amount of data you need to display is larger than the area of the application available.

Notes

The object is an XmText within an XmScrolledWindow. In other words, when you select Scrolled Text from the Palette, two widget instances are created: Text (widget you selected) and Window (parent of Scrolled Text).

Scrolled Window

Description

The XmScrolledWindow widget allows the user to view a portion of a much larger area. XmScrolledWidget combines one or more Scroll Bar widgets and a viewing area to function as a visible window onto some other data display. The visible part of the window can be scrolled through the larger display by the use of Scroll Bar widgets. Scrolling can be automatic. The programmer may control scrolling by setting scrolling policy to user defined and using callbacks.

Notes

- Displays two ScrollBars which may be manipulated to move the area into view.
- May also be configured to permit the application to do the scrolling, which may be more efficient for some applications.

Selection Box

Description

The XmSelectionBox widget is a general Dialog widget that allows the user to select one item from a list. A Selection Box includes a scrollable list, an editable text field for the selected item, labels for the list and text field, and three buttons for control. The user can select an item by scrolling through the list and selecting the desired item or by entering the item name directly into the text edit area. Selecting an item from the list causes that item to appear in the selection text edit area.

Notes

- Displays a scrolling list of items.
- Displays an editable text field for the selected item.
- Displays modifiable labels.
- Displays several buttons for indicating the selection.

Separator

Description

The XmSeparator widget is a primitive widget that visually separates items in a display, that is, separates visually-distinct areas of your application's interface.

Notes

- Several different line drawing styles are provided in horizontal or vertical orientation.
- Supports several styles of line drawing which offer different etch or shadow effects.

SpinBox

Description

The XmSpinBox widget allows the user to choose a value from a circular list of choices. Only one value is displayed at a time. The widget always has two arrows, one for increment, one for decrement, and one or more other children. the list choices are displayed in a text child (XmText or XmTextField).

Notes

- List items can be either textual or numeric.
- Arrow placement is configurable.

Text

Description

The XmText widget is a single-line and multi-line text editor. It can be used for single-line string entry, data entry into a form (with verification procedures), and full-window editing. Text provides the application with a consistent editing system for textual data. Keyboard actions are defined for primary selection, and cutting, pasting, insertion, and deletion of text.

Notes

- The user can edit text in-line.
- Text also supports Motif Drag and Drop.

Text Field

Description

The XmTextField widget creates a single-line string entry area, and provides a single-line text entry field. It has many of the Text widget's resources with the exception of those relating to multiple line text editing. Callbacks notify the application of cursor movement and changes in the text or input focus.

Notes

- Use for single line text entry (more efficient than using a Text widget in single line mode).
- Text may be edited in-line by the user.
- Supports Motif Drag and Drop.

Toggle Button

Description

The XmToggleButton widget allows the user to set an on/off choice, and sets state data within an application. This widget consists of text or a pixmap with an optional indicator.

Notes

- Visual capabilities are like those of the XmLabel.
- May display an indicator that shows the type of choice the user may make.
- Several Toggle Buttons may be grouped together in a Radio Box to support 1-of-N selection.

ViewKit ObjectPak Vk Classes

ViewKit ObjectPak classes are added to the Palette when you select ViewKit as the current language. Initially, ViewKit ObjectPak classes are grouped in the following folders:

- Dialogs

  VkGenericDialog, VkFileSelectionDialog, VkPromptDialog, VkQuestionDialog, VkInfoDialog, VkWarningDialog, VkErrorDialog, VkFatalErrorDialog, VkProgressDialog, VkBusyDialog, and VkInterruptDialog.
- Menus

  Menu Bar, Sub Menu, Option Menu, Popup Menu, Radio Sub Menu, Help Pane, Menu Action, Menu Confirm First Action, Menu Undo Manager, Menu Toggle, Menu Label, and Menu Separator.
- Components

  VkSimpleWindow, VkWindow, VkTabbedDeck, VkTabPanel, VkTabSet, VkRepeatButton, VkOutline, VkCompletionField, VkGraph, VkPie, VkVuMeter, and VkTickMarks.

The following sections provide descriptions of each ViewKit ObjectPak class in alphabetical order (refer to the ICS *ViewKit ObjectPak Programmer's Guide* for more detailed information):

Busy Dialog

Description

The VkBusyDialog class supports a busy dialog (also called a working dialog in OSF/Motif) that is displayed when the application is busy. Used by the VkApp object to display a busy dialog when you place the application in a busy state.

Notes

- By default, does not display any buttons because the busy dialog is intended to lock out user input during a busy state.
- Does not provide any additional functions other than those offered by the VkDialogManager.

Completion Field

Description

The VkCompletionField class provides a text input field component that supports name expansion. While typing in the field, if the user types a space, then the component attempts to complete the current contents of the field based on a list of possible expansions provided by the application.

Notes

Derived from VkComponent.

Error Dialog

Description

The VkErrorDialog class supports standard OSF/Motif error dialogs. Use to inform the user of an invalid action (such as entering out-of-range data) or potentially dangerous conditions (such as the inability to create a backup file).

Notes

- Messages required in the error dialogs should not require any decision by the user.
- Does not provide any additional functions other than those offered by the VkDialogManager

Fatal Error Dialog

Description

The VkFatalErrorDialog class supports an error dialog that terminates the application when the user dismisses it. Use for those errors from which your program cannot recover, such as when an application terminates because it cannot open a necessary data file.

Notes

- Messages required in the error dialogs should not require any decision by the user.
- Does not provide any additional functions other than those offered by the VkDialogManager

Generic Dialog

Description

The VkGenericDialog class provides a convenient interface for creating custom dialogs that use the ObjectPak interface. Custom dialogs that you derive from this class automatically support caching and all other features supported by VkDialogManager. Creating an instance of this class automatically creates a subclass of VkGenericDialog.

Notes

- An abstract subclass of VkDialogManager.
- You can post and manipulate your custom dialogs using the functions provided by VkDialogManager.
- By default, ObjectPak dismisses your dialog when the user clicks on either the OK or Cancel button and continues to post the dialog when the user clicks on Apply.

Graph

Description

The VkGraph class displays and manipulates complex arc-and-node graphs. The graph can be disconnected and contain cycles.

Notes

- Can arrange the nodes horizontally or vertically.
- Can change the orientation interactively.
- Provides controls for interactive zooming, node repositioning, and node alignment.

Help Pane

Description

The VkHelpPanel class provides a simple user interface to a help system. A subclass of VkSubMenu, automatically provides five standard menu items. You must link an external help system to your application to use Help Pane.

Notes

- You can create a VkHelpPane object and add it to another menu.
- You can use the functions provided by VkSubMenu to add custom Help menu items and delete predefined Help menu items.

Info Dialog

Description

The VkInfoDialog class supports standard OSF/Motif information dialogs. Use to display useful information, except error messages.

Notes

- Messages required in the error dialogs should not require any decision by the user.
- Does not provide any additional functions other than those offered by the VkDialogManager.

Interrupt Dialog

Description

The VkInterruptDialog class supports an interruptible busy dialog that you can substitute for the normal busy dialog. Posts a dialog that includes a Cancel button that the user can click on to cancel the current action.

Notes

- You are responsible for cleanup operations required by your application if the user interrupts a process before it is finished.
- Do not directly post and unpost the interruptible busy dialog.

Menu Action

Description

The VkMenuAction class provides a selectable menu item that performs an action. Implemented as a PushButtonGadget. Associated with a callback function that performs an operation and, optionally, a callback function that undoes the operation.

Notes

Provides public functions in addition to those implemented by VkMenuItem.

Menu Bar

Description

The VkMenuBar class provides a menu bar designed to work with VkWindow and some member functions for installing a VkMenuBar object as a menu bar.

Notes

Supports all functions provided by VkMenu class.

Menu Confirm First Action

Description

The VkMenuConfirmFirstAction class provides a selectable menu item that performs an action. When the user selects this menu item type, the application posts a question dialog requesting confirmation. Performs the action only when user confirmed.

Notes

- Derived from VkMenuAction.
- Intended for irrecoverable actions, such as deleting a file.
- Does not support undo callback functions.

Menu Label

Description

The VkMenuLabel class provides a non-selectable label as a menu item. Implemented as a LabelGadget.

Notes

Does not provide any public functions other than those implemented by VkMenuItem.

Menu Separator

Description

The VkMenuSeparator class provides a non-selectable separator as a menu item, and is implemented as a SeparatorGadget.

Notes

Does not provide any public functions other than those implemented by VkMenuItem.

Menu Toggle

Description

The VkMenuToggle class provides a two-state toggle as a menu item. Exhibits simple checkbox behavior unless you add a group of toggles to a VkRadioSubMenu object to enforce radio behavior.

Notes

- Derived from VkMenuAction.
- Provides functions for setting and retrieving the toggle state in addition to the public functions provided by VkMenuItem.

Menu Undo Manager

Description

The VkMenuUndoManager class provides an easy method for users to undo commands that they issue to your application. You add a single menu item to one of your application's menus to create a user interface to Undo Manager. You can use the undo manager to support undoing any command, whether the user issues the command with a menu or with other interface methods.

Notes

- By default, provides multi-level undo support.
- Keeps commands on a stack.
- Must exist in an application if undoCallback is set on any menu item.

Option Menu

Description

The VkOptionMenu class supports option menus that can be used anywhere in your interface.

Notes

Automatically visible when created.

Outline

Description

The VkOutline class displays a textual outline and automatically indents items according to their depth in the outline.

Notes

- Derived from VkComponent.
- If space is insufficient to display the entire outline, automatically displays a scrollbar.
- Displays control icon to the left of each outline item that contains sub-items.

Pie

Description

The VkPie class displays data as a pie chart.

Notes

- Derived from VkMeter.
- Can be fixed size or attempt to resize itself dynamically as it requires more or less room to display the items it contains.

Popup Menu

Description

The VkPopupMenu class supports popup menus that you can attach to one or more widgets in your application. Will pop up automatically when the user clicks on any of those widgets with the right mouse button.

Notes

- Has four different constructors.
- Can also pop up the menu programmatically if you have not attached the popup menu to a widget.

Progress Dialog

Description

The VkProgressDialog class displays a progress meter with a percent complete scale.

Notes

Derived from VkDialogManager.

Prompt Dialog

Description

The VkPromptDialog class supports standard OSF/Motif prompt dialogs that allow the user to enter a text string. Use when you must prompt the user to enter a single piece of information.

Notes

By default, displays only the OK and Cancel buttons. Displays the Apply button only when you provide a callback for that button.

Question Dialog

Description

The VkQuestionDialog class supports standard OSF/Motif question dialogs that allow the user to select among simple choices by clicking on push buttons.

Notes

- By default, displays only the OK and Cancel buttons. Displays the Apply button only when you provide a callback for that button.
- Does note provide any additional functions other than those offered by VkDialogManager.

Radio Sub Menu

Description

The VkRadioSubMenu class supports pull-down menu panes, and intended to support one-of-many collections of VkToggleItem objects. Use as menu panes within a menu bar, or as cascading, pull-right menu in a popup or other pull-down menu.

Notes

- Derived from VkSubMenu.
- Can add to any VkMenuBar, VkPopupMenu, or VkSubMenu by calling their addRadioSubmenu() member functions.

Repeat Button

Description

The VkRepeatButton class provides an auto-repeating push button. Activates when the user clicks on the push button, and begins repeating at a given interval. Deactivates when the user releases the push button.

Notes

- Constructor takes three arguments.
- Derived from VkComponent.

Simple Window

Description

The VkSimpleWindow class supports a top-level window that does not include a menu bar along the top of the window and creates a popup shell as a child of the invisible shell created by your application’s instance of VkApp. Also creates an XmMainWindow widget as a child of the popup shell. Sets up various properties on the shell window and provides simple hooks for window manager interactions.Registers its windows with the application’s VkApp instances to support application-wide services (for example, setting the cursor for all application windows).

Notes

All top-level windows in a ViewKit application must be instances of VkSimpleWindow, VkWindow, or a subclass of one of these classes.

Note: The Make Class dialog is automatically displayed when you click on Simple Window from the Palette.

Sub Menu

Description

The VkSubMenu class supports pull-down menu panes that can be used within a menu bar, or as a cascading, pull-right menu in a popup or other pull-down menu. You can add a sub menu to any type of menu by calling the menu's addSubmenu() member function.

Notes

- Typically does not require instantiation.
- Provides additional public member functions.

Tabbed Deck

Description

The VkTabbedDeck class is a manager class that displays only one child at a time. Each child has an associated tab. Selecting a tab displays the associated child. Set the Generate VkTabbedDeck Source Code toggle on the Classes tab of the ViewKit Generation Preferences panel (Browser:Options:Code Generation Preferences) to cause Builder Xcessory to generate this code. See *"Toggle options"* on page 73 for more detailed information.

Notes

- This class was added to ViewKit as of version 1.3.
- Build Xcessory provides source code to this class for users of ViewKit prior to version 1.3.
- This class has been depricated by VkTabSet as of version 1.5

Tab Panel

Description

The VkTabPanel class displays a row or column of overlaid tabs horizontally. The tab can contain text, a pixmap, or both. Allows the user to click on a tab with the left mouse button to select the tab.

Notes

- Derived from VkComponent.
- One tab is always selected and appears above the others.
- You can register callback functions to perform actions based on the tabs selected.
- This class has been depricated by VkTabSet as of version 1.5

Tab Set

Description

The VkTabSet class is a manager class that displays only one child at a time. Each child has an associated tab. Selecting a tab displays the associated child. .

Notes

- This class was added to ViewKit as of version 1.5.
- Ment to replace VkTabPanel and VkTabDeck..

Tick Marks

Description

The VkTickMarks class displays a vertical set of tick marks. Most frequently used next to a vertical Motif XmScale(3) widget. By default, right-justifies its tick marks and displays its labels to the left.

Notes

- Derived from VkComponent.
- Can be configured to left-justify its tick marks and display its labels to the right.

VuMeter

Description

The VkVuMeter class is a component for displaying a segmented meter and presents a vertical set of segments as a meter display.

Notes

- Value ranges from 0 to 110, with 0 showing the most segments and 110 showing the least.
- Can have a fixed size or attempt to resize itself dynamically.

Window

Description

The VkWindow class supports a top-level window that includes a menu bar along the top of the window and creates a popup shell as a child of the invisible shell created by your application's instance of VkApp. Also creates an XmMainWindow widget as a child of the popup shell. Sets up various properties on the shell window and provides simple hooks for window manager interactions. Registers its windows with the application's VkApp instances to support application-wide services (for example, setting the cursor for all application windows).

Notes

- Derived from VkSimpleWindow.
- All top-level windows in a ViewKit application must be instances of VkSimpleWindow, VkWindow, or a subclass of one of these classes.

Note: The Make Class dialog is automatically displayed when you click on Window from the Palette.

- The subclass created in Builder Xcessory includes several common menu entries in the menu bar, which you can delete or change as necessary.

Warning Dialog

Description

The VkWarningDialog supports standard OSF/Motif warning dialogs. Use to warn the user of the consequences of an action, such as warning the user that an action will irretrievably delete information.

Notes

- By default, the dialogs contain only an OK button.
- Does not provide any addition functions other than those offered by the VkDialogManager.

EnhancementPak Xi Widgets

BX PRO users By default, the BX PRO Palette includes the EnhancementPak widgets.

Builder Xcessory users Start with EPak Widgets must be set on the Behavior tab of the User Preferences dialog to display the EnhancementPak icons. See *"Behavior toggle options"* on page 97 for more detailed information.

EnhancementPak widgets are displayed on the Palette for all platforms and for all languages except Java. The EnhancementPak widgets are grouped in the following folders:

- EPak Primitives

  XiColorSelector, XiCombinationBox, XiExtended18List, XiFontSelector, XiIconButton, XiPanner, XiDataField, and XiPixmapEditor.
- EPak Containers

  XiTree, XiIconBox, XiOutline, XiPaned, XiPorthole, XiStretch, XiTabStack, XiToolbar, XiButtonBox, and XiColumn.
- EPak Graphs

  XiBarPlot, XiCallbackPlot, XiErrorPlot, XiFadePlot, XiHighLowPlot, XiHistoPlot, XiImagePlot, XiLinePlot, XiPiePlot, XiTextPlot, and XiPlotter.

The following sections provide descriptions of each EnhancementPak widget in alphabetical order. Refer to the *EnhancementPak for OSF/Motif Programmer's Reference* and the *ICS GraphPak for OSF/Motif Programmer's Reference* manuals for more detailed information about these widgets.

Button Box

Description

The XiButtonBox widget positions children in a single row or column, with control over their spacing and sizing behavior.

Notes

- Attempts to preserve even spacing between objects
- Can also stretch its children to fill the available space

Color Selector

Description

The XiColorSelector widget allows the user to choose a color value by setting RGB values or selecting a name from a list of available colors.

Notes

The name or color value is dynamically displayed to the user.

Column

Description

The XiColumn widget displays its children stacked in a column, each with an optional associated label. Useful for displaying labeled data-entry fields.

Notes

- Offers several constraint resources that allow specification of characteristics of the label.
- Offers several resources for setting defaults for children that specify no specific values.

Combination Box

Description

The XiCombinationBox allows the user to select an item or items from a list of choices, or enter new values in the text field portion.

Notes

Displays a type-in area showing the current value and a popup list containing other possibilities.

Data Field

Description

The XiDataField widget handles display and entry of data as text, and intended for data entry applications.

Notes

A subclass of Motif XmTextField.

Internationalized Extended List

Description

The XiExtended18List widget allows you to display multi-column data and retrieve the selection.

Notes

- Displays a title for each column
- Displays the data so that columns are visibly separate
- Supports sorting on a particular column
- Supports searching through the data

Font Selector

Description

The XiFontSelector widget allows the user to choose a font by selecting the characteristics of the font.

Notes

Dynamically calculates the set of available fonts that match the specified characteristics.

Icon Box

Description

The XiIconBox widget allows you to position children in a grid at arbitrary cell locations.

Notes

Resizes each child to be the same size.

Icon Button

Description

The XiIconButton widget displays both simple text and a pixmap in several possible configurations.

Notes

Use when you want an object that has both the semantics of a Push Button or Toggle Button and the capability of displaying both simple text and a pixmap.

Outline

Description

The XiOutline widget shows the relationship of objects in a graphical, indented format.

Notes

- Each child is displayed as an entry in an outline, at various levels of indentation.
- User can open and close sub-trees to show or hide data.

Paned

Description

The XiPaned widget allows the user to resize various portions of the application interface, either horizontally or vertically.

Notes

Each child object that the Paned Window controls is associated with a sash that the user may move to change the size of the child.

Panner

Description

The XiPanner widget allows the user to scroll a third widget in two dimensions (easier than using two ScrollBars).

Notes

- Typically used in combination with the Porthole.
- The Panner’s slider shows the location of the visible region of the third widget.
- The user moves the slider to move the visible region.

Pixmap Editor

Description

The XiPixmapEditor widget implements a simple editor for creating and modifying multicolor images.

Notes

- The Pixmap Editor is a simple color pixmap editor bundled into one widget.
- Includes operations for the drawing of simple shapes, and the movement and copying of areas.

Porthole

Description

The XiPorthole widget allows the user to scroll a third widget in two dimensions.

Notes

- Easier to use than using two ScrollBars.
- Usually used in combination with the Panner.
- Manages a single child, and displays a clipped portion of that child within its own bounds.

Stretch

Description

The XiStretch widget allows the user to dynamically change the size of the Stretch widget's single child by dragging the border.

Notes

Changes the size similar to how the window manager resizes a top-level window.

Tab Stack

Description

The XiTabStack widget manages a group of widgets such that only one widget in the group is visible at a time. Each child is associated with a tab that displays text and/or a pixmap. The user selects the tab, interactively determining which child is displayed. Tabs can be configured to appear above, below, and right or left of the work area, with the text oriented in any of the four cardinal directions.

Notes

- Allows user to select tabs, either by pointer or by keyboard traversal.
- When a tab is selected, the tab appears to be raised above the other tabs and the child associated with the tab is made visible.
- One tab is selected at all times.

Toolbar

Description

The XiToolbar widget manages groups of child widgets in either a single row or column. Any type of widget can be a Toolbar item, but the most common is the XiIconButton.

Notes

Supports the display of a popup label over each child widget.

Tree

Description

The XiTree widget shows the relationship of objects in a graphical tree format.

Notes

- Each child is displayed as part of a branch of a tree, at various depths.
- The user can open and close sub-trees to show or hide data.

Java Awt Objects

Java Awt[1] objects are displayed on the Palette when you select Java as the current language. Palette Java objects are grouped in the following folders:

- Java AWT Primitives

  Objects which cannot receive children: AwtButton, AwtLabel, AwtCheckbox, AwtChoice, AwtList, AwtScrollBar, AwtTextArea, AwtTextField, AwtCanvas, and AwtFileDialog.
- Java AWT Containers

  Objects which can receive children: AwtFrame, AwtPanel, AwtApplet, and AwtScrollPane.
- Java AWT Menus

  Objects used to build menus (menus can be contained within the Frame object only): AwtMenuBar, AwtMenu, AwtPopuMenu, AwtMenuItem, AwtCheckboxMenuItem, and AwtMenuSeparator.

Note: Only Java AWT objects are available on the Palette when you select Java as the current language.

Code generated objects

In some cases, non-visual objects are required to set resources. The following objects are not on the Palette, but are created when Builder Xcessory generates code:

- BorderLayout
- FlowLayout
- CardLayout
- GridLayout
- GridBagLayout
- GridBagConstraints
- CheckBoxGroup

These objects supply additional resources in the Resource Editor for the associated object.

The following sections provide descriptions of each Java object on the Palette, in alphabetical order. Refer to your Java AWT documentation for more detailed information.

1. Abstract Window Toolkit (AWT) is a portable GUI library for stand-alone applications and/or applets. Refer to Java documentation for more detailed information.

Applet

Description

The AwtApplet object is a top level panel for applications designed to run within a Web browser (unlike a frame that pops up a separate window outside the Web browser)

Notes

- Subclassed from panel.
- Additional methods automatically called from browser.
- Can be run from a browser or Applet viewer.
- Default is FlowLayout.

Button

Description

The AwtButton class is a GUI button for which you can define an action that will happen when the button is pressed. Consists of a text label or pixmap surrounded by a border shadow.

Notes

When the button is selected, the object appears as if it has been pressed in.

Canvas

Description

The AwtCanvas class is general purpose component that allows you to paint or trap input events from the user.

Notes

- Can be used for creating graphics.
- Can be used as a base class that can be subclassed in order to create your own custom components.

Checkbox

Description

The AwtCheckbox class is a component that has an on/off state. Generates an event when state changes.

Notes

Maintains a Boolean state (whether checked or unchecked).

Checkbox Menu Item

Description

The AwtCheckboxMenuItem class is a checkbox with a text label in a GUI menu.

Notes

Maintains a Boolean state (whether checked or unchecked).

Choice

Description

The AwtChoice object is a menu of options that drops down from a button. The button displays an icon to indicate that a menu is available. Generates an event when a menu item is selected.

Notes

The button displays the currently selected option as its label.

Dialog

Description

The AwtDialog object is a top-level window used to create dialogs. Can be modal, that is, will allow only this window to receive input from the user.

Notes

- Dialog can have a title.
- Dialog can be resizable.
- Default layout is BorderLayout.

Note: The Make Class dialog is automatically displayed when you click on Dialog from the Palette.

File Dialog

Description

The AwtFileDialog object is a dialog that uses the native file chooser dialog to select a file from the file system

Notes

You can control which files are displayed in the dialog.

Note: The Make Class dialog is automatically displayed when you click on File Dialog from the Palette.

Frame

Description

The AwtFrame object is an optionally resizable top-level window that has a border and can also have an associated menu bar, title, cursor, or icon.

Notes

- Pops up a separate window outside the Web browser.
- When the Frame is no longer necessary, the dispose method must be called to allow the frame to release its window system resources for reuse.
- Default layout is BorderLayout.

Label

Description

The AwtLabel object is a component that displays a string at a certain location.

Notes

Constant values specify the text alignment within the component.

List

Description

The AwtList object scrolls strings. You can set the number of visible rows in the list and you can set whether the selection of more than one item is allowed.

Notes

- Strings can be added and removed from List.
- You can query the list.
- When the user double-clicks on an item, a List generates an event.

Menu

Description

The AwtMenu object contains a pulldown menu pane that appears when selected. Displays menu items.

Notes

Child of a menu bar.

Menu Bar

Description

The AwtMenuBar class is a menu bar that may be displayed within a Frame.

Notes

Can only be the child of a Frame.

Menu Separator

Description

The AwtMenuSeparator is used to separate new items in a pulldown menu.

Menu Item

Description

The AwtMenuItem is a menu item with a specific text label and can be added to a menu pane.

Panel

Description

The AwtPanel object is a container that can be used inside other containers. Allows you to make more intricate layouts by combining them with subpanels. It also can be subclassed to create custom containers.

Notes

- Does not create a separate window shell of its own.
- You can nest Panels to achieve a specific layout.
- Default layout is FlowLayout.

Popup Menu

Description

The AwtPopupMenu object displays popup menus and is used like AwtMenu.

Notes

Subclass of Menu.

Scrollbar

Description

The AwtScrollbar object is a scrollbar that can be used to build scrollable canvases.

Scroll Pane

Description

The AwtScrollPane container contains a single child component. Displays a fixed-size area of the child and provides horizontal and vertical scrollbars so the user can scroll the child component within the view of the Scroll Pane.

Notes

The child that Scroll Pane contains is usually larger than the Scroll Pane itself.

Text Area

Description

The AwtTextArea object is a simple text editing component.

Notes

Allows viewing and optional editing of multiline text.

Text Field

Description

The AwtTextField object is a single-line text component that can be used to build forms.

SGI Sgm Widgets

Silicon Graphics Incorporated (SGI) Sgm widgets are included on the Palette when you run Builder Xcessory on an SGI platform. The following sections provide descriptions of each SGI widget in alphabetical order (refer to your SGI documentation for more detailed information):

Column

Description

The SgColumn widget displays its children stacked in column with an optional associated label.

Notes

- Useful for displaying labeled data-entry fields.
- Offers several resources to set various characteristics of the label.

Dial

Description

The SgDial widget allows the user to specify or change a value in a given range. Users change the current value by direct manipulation of the dial (dragging or clicking on the appropriate tick mark that represents the desired value). The appearance and the behavior of the dial can be modified. For example, the angular range in degrees through which the dial is allowed to rotate and the color of the dial and tick marks can be changed.

Notes

- Use dials as an alternative to scales for setting parameters. Dials are best for numeric parameters where the range of allowable values is small and the values are discrete.
- In a group of dials, place each dial label in the same position relative to its dial (that is, either all the labels should be below the dials, or all the labels should be above the dials).

Drop Pocket

Description

The SgDropPocket widget displays the desktop icon that represents the current file or directory that has been dropped on it. Fully implements drag and drop protocol with the Indigo Magic™ desktop.

Notes

The user can drop a desktop icon into the drop pocket to automatically update data in an application.

Dyna Menu

Description

Use the SgDynaMenu widget for entries that are not always available. Entries are placed in a menu and disabled when they are not available.

Notes

Because menu entries are visible only when the application is in a specific state, dynamic menu entries do not allow the user to learn what entries are in each of the menus.

Finder

Description

The SgFinder widget allows users to find an application or file using drag and drop or by entering the application or file name. Combines a Drop Pocket with a text field and a Dyna Menu to hold recent entries.

Notes

Useful for allowing a user to quickly change the file system component (file, directory, and so forth) that the application is operating on.

Graph

Description

The SgGraph widget allows you to display any group of widgets as a graph, with each widget representing a node. The graph can be disconnected, as well as contain cycles. Arcs used to connect the nodes are instances of an SgArc widget, developed specifically for use with the SgGraph widget. Can arrange all nodes either horizontally or vertically according to an internal layout algorithm, and supports an edit mode in which arcs and nodes may be interactively repositioned as well as created.

Notes

- Arcs may be undirected, directed, or bi-directed.
- Does not understand arc direction semantics (for layout and editing purposes, an arc always has a parent and a child regardless of direction).
- In read-only mode, all events are passed directly to its children.
- In edit mode, SgGraph takes all device events for editing commands.

Grid

Description

SgGrid is a container widget that has no input semantics of its own. Arranges its children in a two dimensional grid of arbitrary size. Each row and column of this grid may be separately designated as having a fixed size or as having some degree of stretchability. Each child may be resizable in either or both directions, or forced to a fixed size. If a child is a fixed size, and smaller than the cell that contains it, the child's position within the cell is determined by an XmNgravity resource.

Notes

- The position of each child must be set using the XmNrow and XmNcolumn resources. If no position is specified, the child is placed in an unspecified free cell.
- The resizability of each row and column must be set using convenience functions SgGridSetRowResize and SgGridSetColumnResize. The default is for all rows and columns to be resizable. All widgets are resized according to their relative natural size.
- Unmapping a child does not affect Grid.

Horizontal Paned Window

Description

SgHorizontalPanedWindow is a composite widget that lays out children in a horizontally tiled format and allows the user to horizontally resize various portions of the application interface. The first child is inserted at the left side of the Horizontal Paned Window and each successive child is inserted to the right.

The Horizontal Paned Window grows to match the height of its tallest child and all other children are forced to this height. The width of the Horizontal Paned Window is equal to the sum of the widths of all its children, the spacing between them, and the size of the left and right margins.

Notes

- Each child object that the Horizontal Paned Window controls is associated with a sash that the user can move to change the size of the child.
- Various constraint resources control the resizing behavior of children when the Horizontal Paned Window is resized.

IRIS GL Draw or Open GL Draw

Description

Depending on whether you are using the IRIS GL proprietary 3-D graphics standard or the Open GL non-proprietary 3-D graphics standard, the following widget is displayed on the Palette:

- GlxMDraw—Displayed when running the IRIS GL standard. This is the default, and you can also explicitly set it with the `+openGL` flag on the `bx` command line.
- GLwMDrawingArea—Displayed when running the OpenGL standard. You can also explicitly set it with the `-openGL flag` on the `bx` command line.

Note: Although these widgets are not actual SGI Sgm widgets, they are included with the SGI Sgm widgets for your convenience.

Notes

- Note the state of the openGL flag is preserved between sessions.
- If you are unsure of which object to load and use, the Open GL Draw is recommended. The IRIS GL Draw object is really only included for backwards-compatibility purposes.
- OpenGL and IRIS GL cannot be used in the same application.

Icon Gadget

Description

The SgIconGadget widget displays a labeled pixmap. Does not accept any button or key input, and the help callback is the only callback defined. Receives enter and leave events. Can be displayed in colors other than that provided by its parent, a restriction that limits the usefulness of gadgets in many situation.

Notes

- Can contain both text and a pixmap.
- SgIconGadget text is a compound string.
- The text can be multi-directional, multiline, and/or multi-font.
- When a SgIconGadget is insensitive, its text is stippled, or the user-supplied insensitive pixmap is displayed.

LED Button

Description

SgLEDButton is an SGI enhanced XmToggleButton that allows the user to set an on/off choice. The LED Button adds two new indicator types to those of the standard Motif Toggle Button: Xm3D_N_OF_MANY and Xm3D_ONE_OF_MANY. These types cause the indicator to be drawn like an LED with on/off state indicated by a "lit/unlit" color selection.

Notes

Apart from indicator types, LED Button is identical to an XmToggleButton.

Rubber Board

Description

SgRubberBoard allows the user to set up initial and final layout states for an interface and interpolates any intervening layouts.

Notes

Exercise caution when using this manager. Adjustments are not made by the Rubber Board for any changes initiated by its children. Therefore, changes to a child's size due to localization, and so forth, are not taken into account when the Rubber Board recalculates child size and location when the Rubber Board itself is resized.

Spring Box

Description

SgSpringBox is a container widget with no input semantics of its own. Arranges its children in a single row or column based on a set of spring constraints assigned to each child. Allows layouts similar to those supported by the XmForm widget, but usually easier to set up. Possible to create some layouts that cannot be achieved with the XmForm widget (for example, centering a column of widgets is very easy to do with the SgSpringBox widget, but nearly impossible using the XmForm).

Notes

- Each child of an SgSpringBox widget has six constraints associated with it.
- The resources XmNverticalSpring and XmNhorizontalSpring control the degree of "springiness" in each child. A value of zero means the child cannot be resized in that direction. For non-zero values, the values are compared to the values of other springs in the overall system to determine the proportional effects of any resizing. The default value of both resources is zero.
- Each child has a spring between its left, right, top, and bottom sides and whatever boundary it is adjacent to.

Thumb Wheel

Description

The SgThumbWheel widget allows users to specify or modify a value from within a range of values or from an infinite range. Users change the current value by clicking and dragging (spinning) the wheel. Can also include a home button that returns the thumbwheel to a default value. Thumbwheels can be oriented horizontally or vertically.

Notes

- Use to change the values of continuous variables.
- Use with finite ranges for zooming operations.
- Use with an infinite range for rotating objects.
- Use when screen real estate is limited.
- Update the object or value as the user moves the thumbwheel to imply direct, continuous manipulation.

Visual Draw

Description

The SgVisualDraw widget allows its X Visual to be set such that graphics may be rendered using a visual different from the surrounding widget.

Notes

Almost identical to the Motif Drawing Area widget.

CDE Dt Widgets

Common Desktop Environment (CDE) widgets are included on the Palette when you run Builder Xcessory on any platform with CDE and Builder Xcessory CDE support (except on SunOS platforms). The following sections provide descriptions of each CDE widget in alphabetical order (refer to your CDE Programmer's Guide for more detailed information):

Combination Box

Description

The DtComboBox widget combines a value display field and a list box in a control that displays one of many valid choices for the text field.

Notes

The value display can be editable or not editable, which restricts the user options to the contents of the popup list

Editor

Description

The DtEditor widget supports creating and editing text files, and provides a consistent method for editing text data to applications running in the desktop environment.

Notes

Very useful for providing text edit functions in an application that also integrates automatically with the CDE Drag and Drop model.

Help Dialog

Description

The DtHelpDialog widget provides users with the capability for viewing and navigating structured online help information (CDE help volumes). Functionality includes text and graphics rendering, embedded hypertext links and various navigation methods to move through online help information. Supports rendering of CDE help volumes, system manual pages, text files, and character string values.

Notes

Use this dialog when displaying the complete help system for an application that is to be integrated onto the CDE desktop.

Help Quick Dialog

Description

The DtHelpQuickDialog widget provides users with a constrained set of functionality for viewing and navigating structured online information (CDE help volumes). Functionality includes text and graphics rendering, embedded hypertext links and limited navigation methods to move through online help information. Supports rendering of CDE help volume, system manual pages, text file, and character string values.

Notes

Use this dialog for context sensitive help items.

Menu Button

Description

The DtMenuButton widget is a command widget that provides menu cascading capability outside of a menu pane.

Notes

- Complements the menu cascading functionality of an XmCascadeButton widget.
- Can only be instantiated outside a MenuBar, Pulldown, or Popup (use XmCascadeButton widget inside a MenuPane.)

Spin Box

Description

The DtSpinBox widget is a user interface control used to increment and decrement an arbitrary text or numeric field. Use to progress through a list of text items, for example, to cycle through the months of the year or days of the month, or increment and decrement a numeric entry.

Notes

DtSpinBox is a subclass of the XmManager class and is used to display a text field and two arrows.

Terminal Emulator

Description

The DtTerm widget, part of the runtime environment, is a window that behaves as a terminal, enabling access to traditional terminal-based applications from within the desktop. Provides the functionality required to emulate an ANSI X3.64-1979-style terminal emulator (specifically a DEC VT220-like terminal with extensions).

Notes

- Use Common Desktop Environment Motif widgets to add display enhancements to a terminal emulator in your application, such as pop-up menus and scroll bars.
- Widget library, libDtTerm, provides DtTerm widget for use in adding a terminal emulator window to a GUI

License Manager

C

Note:

Builder Xcessory is licencsed under two mechanisms, one for UNIX systems and another for Linux based systems. The UNIX licensing mechanism is based on FlexLM, and is documented in this chapter. The Linux licensing mechanism is documented on the Linux distribution CD, in the file “license.lin”.

Overview

This appendix describes how to start and use the License Manager, and includes the following sections:

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index