



# *Builder User's Guide*

COPYRIGHT NOTICE ON THE VERSION 5.0 SOFTWARE

©1990 - 2000 Applix, Inc. All Rights Reserved.

003-BLDU-01-E-5.0

Applix, Inc. prepared the information contained in this document for use by Applix personnel, customers, and prospects. Applix reserves the right to change the information in this document without prior notice. The contents herein should not be construed as a representation or warranty by Applix. Applix assumes no responsibility for any errors that may appear in this document.

The Proximity Thesauri ®

©1985 Merriam-Webster Inc.

©1988 Williams Collins Sons & Co. Ltd.

©1989 Van Dale Lexicografie bv. ©1989 Nathan. ©1989 Kruger.

©1989 Zanichelli. ©1989 International Data Education a s.

©1989 C.A. Stromber A B. ©1989 Espasa-Calpe.

©1983-2000. Proximity Technology, Inc.

All Rights Reserved.

The Proximity Linguibase And Hyphenation Systems®

©1983 Merriam-Webster Inc.

©1984, 1985, 1986, 1988, 1990 Williams Collins Sons & Co. Ltd.

©1987, 1989 Van Dale Lexicografie bv. ©1988 Munksgaard International Publishers Ltd.

©1988, 1989 International Data Education a s.

©1983-2000 Proximity Technology, Inc.

All Rights Reserved

©1989-2000 Blueberry Software, Inc.

All Rights Reserved.

The Applix Graphics Filter Pack contains elements of the Generator Metafile Development Libraries (MDL/G)

©1988-2000 Henderson Software, Inc.

All Rights Reserved

©2000 T/Maker Company

Clickart and T/Maker are registered trademarks of T/Maker Company

All Rights Reserved Worldwide

©2000 Gallium Company

FontTastic is a trademark of Gallium Software, Inc.

All Rights Reserved

ImageStream® Graphics and Presentation Filters

© 1991-2000, Inso Corporation

All Rights Reserved

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraphs (c) (1) (ii) of SFARS 252.277-7013, or in FAR 52.227-19, as applicable.

Hardware and software products mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

Applix is a registered trademark of Applix, Inc. Applixware, Applix Real Time, Applix Data, and Applix Builder are trademarks of Applix, Inc.

**This manual was produced using Applixware.**

Printed: February 2000

---

# Contents

---

## Preface

About This Manual .....	xv
Audience .....	xv
Purpose .....	xvi
Conventions Used in This Manual .....	xvii

## Chapter 1     **Introducing Applix Builder**

What is Applix Builder? .....	1-2
Applix Builder Tools .....	1-3
Installing and Licensing Applix Builder .....	1-5
Starting Applix Builder .....	1-5
Starting Applix Builder from Applixware .....	1-5
Exiting Applix Builder .....	1-6
Builder Preferences .....	1-6
Sample Applications .....	1-8

## Chapter 2     **Using the Browser Tool**

Viewing Application Structure .....	2-2
-------------------------------------	-----

---

ExpressLine Icons .....	2-3
Popup Menu .....	2-3
Using the Class Browser .....	2-3
Copying a Method Name .....	2-5
Adding Classes .....	2-5
Editing Classes .....	2-6
Deleting Classes .....	2-9
Saving Classes .....	2-9
Loading Classes .....	2-10
Popup Menu .....	2-13
Application File Options .....	2-14
Opening a New Application .....	2-14
Opening an Application .....	2-15
Saving a New Application .....	2-16
Saving an Application under a Different Name .....	2-19
Sending an Application .....	2-19
Printing Object Sources .....	2-20
Reverting an Application .....	2-20
Deleting an Application .....	2-21
Exiting an Application .....	2-22
Adding and Deleting Objects .....	2-22
Adding an Object .....	2-23
Deleting an Object .....	2-26
Cutting, Copying, and Pasting Objects .....	2-26
Changing Object Properties .....	2-27

---

Setting the Object Method Source.....	2-28
Global Find and Replace .....	2-29

### **Chapter 3 Using the Source Tool**

Using the Data Source Dialog Box.....	3-2
Creating a Data Source.....	3-3
Editing a Data Source .....	3-4
Renaming a Data Source .....	3-5
Deleting a Data Source .....	3-5
Creating a Data Set.....	3-6
Selecting a Database .....	3-6
Selecting a Table .....	3-9
Saving a Query .....	3-13
Starting a New Query.....	3-14
Auto-Query and Query .....	3-14
Setting Query Conditions.....	3-15
Other Query Conditions.....	3-23
Column.....	3-23
Range .....	3-25
Subselect.....	3-25
Multiple Query Conditions .....	3-28
Adding Columns.....	3-34
Column Headings.....	3-36
Group by and Having.....	3-38

---

Table Join . . . . .	3-41
Other Joins . . . . .	3-44
Data Set Joins . . . . .	3-47
SQL Source . . . . .	3-49
Creating a Real Time Data Source . . . . .	3-52
Defining a Template . . . . .	3-54
Modifying a Template . . . . .	3-55
Editing a Record . . . . .	3-57
Field Filters . . . . .	3-61
Entering an RTSQL Query . . . . .	3-62

## **Chapter 4      Using the Designer Tool**

Using the Designer Dialog Box . . . . .	4-2
Creating a Dialog Box . . . . .	4-2
Renaming a Dialog Box . . . . .	4-4
Copying a Dialog Box . . . . .	4-5
Deleting a Dialog Box . . . . .	4-5
Editing a Dialog Box . . . . .	4-6
Changing Dialog Box Settings . . . . .	4-7
Dialog Box Controls . . . . .	4-9
Decorative Elements . . . . .	4-11
Adding a Control . . . . .	4-14
Positioning Controls . . . . .	4-15
Control Attributes . . . . .	4-16

---

Control IDs.....	4-17
Selecting Controls.....	4-17
Character Settings.....	4-19
Color Settings.....	4-21
Deleting Controls.....	4-24
Cut, Copy, Paste, and Paste Dialog.....	4-24
Undo.....	4-26
Creating a Dialog Box with a Data Set.....	4-26
Assigning a Menu Bar.....	4-28
Identifying Parts of the Menus.....	4-29
Starting the Menu Bar Editor.....	4-31
Displaying Menu Items.....	4-33
Displaying Cascading Menu Options.....	4-34
Moving Up through Menu Displays.....	4-35
Displaying Information for a Menu Item.....	4-36
Hidden Menus.....	4-39
Changing Menu Items.....	4-40
Adding Menu Items.....	4-43
Deleting Menu Items.....	4-51
Moving Menu Items.....	4-52
Exiting the Menu Bar Editor.....	4-53
Default Menu Bar.....	4-54
Editing Menu Bar Methods.....	4-54
Creating a Menu Bar Using an Array.....	4-55
An Example Menu Bar Defined Using Arrays.....	4-64

---

Making Menu Options Grayed or Toggled .....	4-65
---------------------------------------------	------

## **Chapter 5 Using the Connector Tool**

Connecting Dialog Box Controls .....	5-2
Choosing a Data Source .....	5-4
None .....	5-4
Macro .....	5-4
Data Set .....	5-5
Real Time .....	5-7
Using a Display Map .....	5-8
Using a Validator .....	5-10
Connecting an Edit Box .....	5-12
Connecting a Table .....	5-15
Creating a Data Set Table .....	5-16
Creating a Real Time Table .....	5-17
Creating a Data Set CrossTable .....	5-19

## **Chapter 6 Using the Debugger Tool**

Starting the Debugger .....	6-2
ExpressLine Icons .....	6-4
Status Area Switches .....	6-4
Popup Menus .....	6-5
Selecting Objects .....	6-8
Setting Break Points .....	6-9

---

Setting Multiple Break Points . . . . .	6-9
Clearing Break Points . . . . .	6-11
Setting Watch List Variables . . . . .	6-11
Using Watch List Variables . . . . .	6-12
Running an Application in the Debugger . . . . .	6-14
Handling Errors . . . . .	6-15
Moving Through the Call Stack . . . . .	6-16
Displaying Variable Values . . . . .	6-17
Execute to Here . . . . .	6-17
Changing Debugging Status . . . . .	6-17

## **Chapter 7      Running and Distributing Applications**

Running Applications from Applix Builder . . . . .	7-2
Running Applications from UNIX/Linux . . . . .	7-2
Before You Run axmain . . . . .	7-2
Running a Macro . . . . .	7-3
Running an Application . . . . .	7-4
Running Multiple Applications . . . . .	7-4
Exiting axmain . . . . .	7-7
Distributing Applications . . . . .	7-8
Making a Macro . . . . .	7-8
Creating a Turbo File . . . . .	7-9
Creating a Distribution File . . . . .	7-10

---

## Chapter 8

### Advanced Topics

Object Libraries . . . . .	8-2
Creating an Object Library . . . . .	8-2
Editing an Object Library . . . . .	8-5
Saving an Object Library . . . . .	8-10
Using an Object Library . . . . .	8-11
External Resources . . . . .	8-15
C++ Class Libraries . . . . .	8-18
Defining the Interface Class and Shared Libraries . . . . .	8-18
Loading C++ Class Libraries . . . . .	8-22
Using C++ Classes In an Application . . . . .	8-23
Distribution and Turbo File Considerations . . . . .	8-29
Remote Objects . . . . .	8-29
Common Dialog Boxes . . . . .	8-32
Applix Builder Bitmaps . . . . .	8-33

## Chapter 9

### CORBA Support

CORBA Architecture . . . . .	9-1
CORBA Terminology . . . . .	9-4
Using CORBA Objects . . . . .	9-4
Using a CORBA Object Reference . . . . .	9-7
Builder Programming Issues . . . . .	9-10
Calling CORBA Methods . . . . .	9-10

---

Invocation Mode .....	9-10
Class Inheritance .....	9-11
Formats and Structs .....	9-12
Data Mapping .....	9-12
Using Builder Objects as Corba Objects .....	9-13
The Applixware CORBA Server Environment .....	9-14
Applix Object Registry .....	9-15
Applixware Object Registration Macros .....	9-15
Callbacks .....	9-18
Creating CORBA Objects in Builder .....	9-19
Generating Files .....	9-22
Coding the CORBA Server .....	9-23

## **Chapter 10      Building User Interfaces**

Designing Dialog Boxes .....	10-2
Dialog Box Guidelines .....	10-2
Form Guidelines .....	10-5
Implementing Applications .....	10-5
Anyware ELF Macros .....	10-6
Menu Bar Programming .....	10-8
Customizing the Iconbar .....	10-11

---

# Figures

---

Figure 4-1 Cascading Menu Options in Menu Bar Editor . . . . .	4-35
Figure 4-2 Moving Up through Menu Displays . . . . .	4-36
Figure 4-3 Menu Bar Array . . . . .	4-55
Figure 4-4 Menu Bar . . . . .	4-61

---

# Preface

---

## About This Manual

This manual is the *Applix Builder User's Guide*. It describes the Applix Builder graphical application development environment and how you can use Applix Builder to create robust, event-driven database or Real Time applications. You can also construct full-featured stand-alone applications, or applications integrated with Applixware applications. This manual explains the features of each Builder tool.

## Audience

The projected audience for the Applix Builder manuals and on-line help are application developers interested in creating graphical, data-centric applications. Applix Builder is a developer's environment for creating applications that are capable of running independent of Applix Builder. The manual assumes the following:

- Entry level programming experience.
- Basic knowledge of object-oriented programming concepts.

## *Audience*

---

- Basic knowledge of Applix ELF (Extension Language Facility).
- Basic knowledge of SQL.

## **Purpose**

The purpose of the manual is to teach you how to create, modify, debug, and run applications using Applix Builder.

## Conventions Used in This Manual

The following typeface conventions are used throughout this manual:

<b>Helvetica</b>	<p>Helvetica text indicates that this option or object appears in the document window. For example, "Type the name of the document in the File name entry area."</p> <p>File names and directories are also indicated by Helvetica text. For example, "This file is located in your axhome directory."</p> <p>Applixware keys are printed in a Helvetica uppercase typeface. For example, "Press the TAB key."</p>
<b>Helvetica Bold</b>	<p>Bold Helvetica text indicates an option to choose or text to type. It usually appears in numbered steps as shown in the following example:</p> <ol style="list-style-type: none"><li>1. Type <b>2.5</b> in the Line spacing entry area.</li><li>2. Click <b>Apply</b>.</li></ol>
<i>Italics</i>	<p>Words are italicized for emphasis or to draw your attention to a new term. For example, "<i>Do not</i> press the RETURN key," or "This action is called <i>word wrapping</i>."</p> <p>Italic type is also used to indicate variable information, as in "Put Applixware in the /user/<i>your_name</i> directory."</p>

Menu Name → Option Name	Whenever you see a reference to a menu option, the option is identified using the following notation:  Menu Name → Option Name  For example, "Choose <b>File</b> → <b>Save</b> ."
OK and Apply	When numbered instructions are included in the text, we omit the final "Click <b>OK</b> or <b>Apply</b> " statement for brevity.

---

# 1 Introducing Applix Builder

---

Applix Builder is a graphical application development environment with object-oriented tools and features. You can use the Builder tools to create custom applications and interfaces for your information. This chapter covers the following topics:

- What is Applix Builder
- Applix Builder tools
- Installing and licensing Applix Builder
- Starting Applix Builder
- Pop up menus
- Builder preferences
- Sample applications

## **What is Applix Builder?**

Applix Builder is a graphical rapid application development environment with object-oriented tools and features. You can use Applix Builder to create a broad range of applications with a minimal amount of coding.

Use Applix Builder to create modular, event-driven applications. You can modify object events in complex applications using object-oriented methods and ELF macros. Builder applications are platform independent, allowing you to run the same application on different hardware and operating system combinations.

With Applix Builder you can:

- create a database application querying multiple databases
- create a Real Time application for monitoring and reacting to time-critical information
- integrate C++ libraries to create powerful, custom applications that leverage existing code base
- create custom, reusable objects shared across multiple applications
- create applications integrated with one or more Applixware applications
- troubleshoot applications with an integrated debugger
- distribute applications to users on various hardware and operating system platforms

Applix Builder tools and features are accessible from the Applix Builder main window. The graphical tools allow you to rapidly create powerful applications in a few steps.



## Applix Builder Tools

The Applix Builder main window is the Browser. The Browser gives you access to all Applix Builder features in addition to allowing you to view the structure of objects in the application. You can also add or delete objects, edit the methods of an object, or view the base classes.

Applix Builder contains several tools for producing applications. The main Applix Builder tools are:

Source	The Source allows you to add, delete, edit, and define Data Sets and Real Time Data Feeds. You can also join Data Sets for your application.
Designer	The Designer allows you to create dialog boxes that provide the interface to data sources or macros in the application. You can create, delete, edit, and design dialog boxes for your application.
Connector	The Connector allows you to connect objects in a dialog box to a data source or macros in the application.
Debugger	The Debugger allows you to debug an application. You can select objects, set break points in object sources, and control the flow of an application through the debugging process.
Run	Click on the Run icon to execute an application from Applix Builder.

Applix Builder also contains a Source Editor that allows you to edit the methods and macros for an object. The Source Editor is accessible from the Connector, Browser, or Object Library dialog box. The Source Editor is also accessible in the Class Browser for user-defined classes.

The Applix Builder tools and the Source Editor are discussed in detail in separate chapters in this manual. Chapter 2, "Object-Oriented Concepts", in the *Applix Builder Programmer's Reference* reviews object-oriented concepts as they relate to Applix Builder.

## Installing and Licensing Applix Builder

Refer to the *Applixware System Administrator's Guide* or the *Applixware Linux System Administrator's Guide* for information on installing and licensing Applix Builder.

## Starting Applix Builder

To start Applix Builder at your operating system prompt:

1. Type **builder**.
2. Press **RETURN**.

The Browser window appears. If you do not start Applix Builder from the directory where it is installed, you must type the full path name of the installation directory with the builder command.

**NOTE:** You must have a Builder license to run Applix Builder.

## Starting Applix Builder from Applixware

If Applix Builder is installed with Applixware you can start Applix Builder from an Applixware application. To start Applix Builder from Applixware choose **★ → Builder** from any application or click on the Builder icon on the Applixware Iconbar.

The Browser window appear.

## Exiting Applix Builder

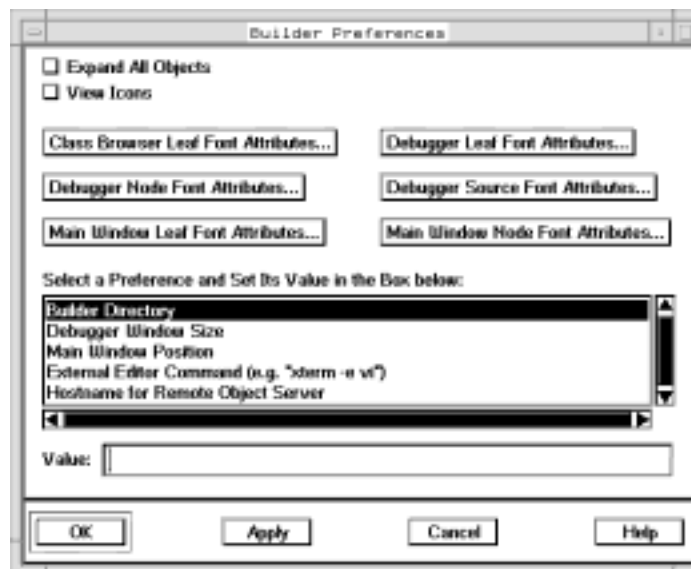
To exit Applix Builder, choose File → Exit from the Browser window. If you made any changes to the application since it was last saved, Applix Builder displays the Exit dialog box. You can save the application, abandon all changes, or cancel the exit.

## Builder Preferences

Use Builder Preferences to customize various aspects of your Applix Builder user environment. Changes you make in the Builder Preferences affect your user environment until you explicitly change them again.

To change Builder Preferences:

1. Choose ✖ → **Builder Preferences** from the Browser window to display the Builder Preferences dialog box.



2. Click on an option and set the default value. See the associated help topic for a detailed description of the Builder preference.

## Sample Applications

Applix Builder provides sample applications that demonstrate the diverse implementation of Applix Builder methods. Sample applications are available in the `/install_dir/axdata/eng/Demos/MainDemo/bld_demo` directory. Each application has a subdirectory that contains source code and all include files for the application.

Choose Tools → Sample Applications to access the sample application directories. The Directory Displayer displays the sample directories in the `/install_dir/axdata/eng/Demos/MainDemo/bld_demo` directory. Read the README file in the application subdirectory for information on using and implementing each sample application.

---

# 2 Using the Browser Tool

---

Use the Browse tool to view the structure of the application and to browse the application objects. This chapter covers the following topics:

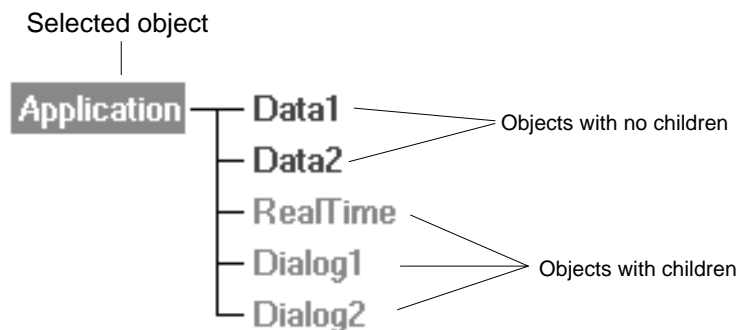
- Viewing Application Structure
- Using the Class Browser
- Application file options
- Adding and deleting objects
- Changing object properties

## Viewing Application Structure

The Applix Builder main window is the Browser. The Browser gives you access to all Applix Builder features in addition to allowing you to view the structure of objects in the application. You can also add or delete objects, edit the methods of an object, or view the base classes.

When the Browser window appears, the Application object appears in the Browser area. All applications have an Application object that other objects descend from. The Application object is selected by default in the Browser window when you start Applix Builder or open a file. Double-click on the Application object to expand the object hierarchy.

Objects that do not have children (leaf) appear green on color displays. Objects that have children (node) appear brown on color displays. Double-click on an object with children to expand the object hierarchy. Double-click on a parent object to contract the object hierarchy. Choose **✳** → Builder Preferences to set leaf or node font attributes.



The hierarchy of the objects in the application is represented by the tree structure in the Browser window. The hierarchy is a relationship hierarchy, not an inheritance hierarchy. Click on an object to select it.

A selected object is highlighted. You can only select one object at a time in the Browser window.

### **ExpressLine Icons**

Use the ExpressLine icons to quickly access menu options in the Browser window.

### **Popup Menu**

A popup menu is available in the Browser window to provide quick access to the Browser menu options. When the mouse pointer is in the Browser window and you press the right mouse button the popup menu appears. If you choose a menu option and release the mouse button the editing action occurs and the popup menu disappears.

## **Using the Class Browser**

Objects can inherit attributes from base classes, either the Builder base classes, or from other, user-defined classes. Choose View → Classes to see the base class objects available in the current application and to add classes to the application. A Class Browser window appears with all the available classes.



Click on a class and all the set and get methods for the class appear in the List area. Choose All Methods in the List radio group to see all methods available to the class, including inherited methods in the higher level classes from which the current class descends. Choose Events in the List radio group to see all events for the current class.

Choose File → Exit to dismiss the dialog box when you are finished looking through the classes and methods.

## Copying a Method Name

You can copy a method name from the List area and paste the method name into an object method source with the **Class → Copy Method** menu option. To copy a method name:

1. Select a method name in the List area.
2. Choose **Class → Copy Method**.

Click in an object method source and choose **Edit → Paste** to paste the method name.

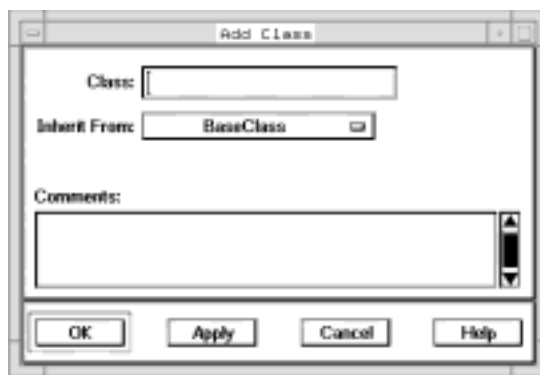
## Adding Classes

Objects in different applications share many of the same features. For example, a `ButtonClass` object to close a dialog box is a common requirement. Instead of creating an object from scratch or inserting a source to obtain object properties, you can create a user-defined class.

A user-defined class can be reused in multiple applications. Changes to a user-defined class are immediately available to applications referencing the class, unless the classes are localized to the application.

To create a user-defined class in the Class Browser window:

1. Choose **Class → Add**. The Add Class dialog box appears.



2. Type the class name in the Class entry area.
3. Choose the class to inherit attributes from with the Inherit From option.
4. Add comments about the class in the Comments edit area.
5. Click **OK**.

The class is added to the application. The class appears in the Class Browser window as a child of the class from which it inherits attributes.

## Editing Classes

After you add a user-defined class you need to define the methods and properties of the class. To define the methods in a user-defined class:

1. Select the class in the Class Browser window.
2. Choose **Class** → **Edit Methods**. An Edit Source window appears.

Add methods to define the actions and properties of objects created with the class. The class method source is loaded with three empty events:

- `class_initialize_event`
- `class_terminate_event`

Use the `class_initialize_event` to define initial class settings. A `class_initialize_event` is called once during application execution, before the `ApplicationClass` object's `initialize_event`. Do not use the `class_initialize_event` to initialize individual object information. Object specific code should be called in the object's `initialize_event`.

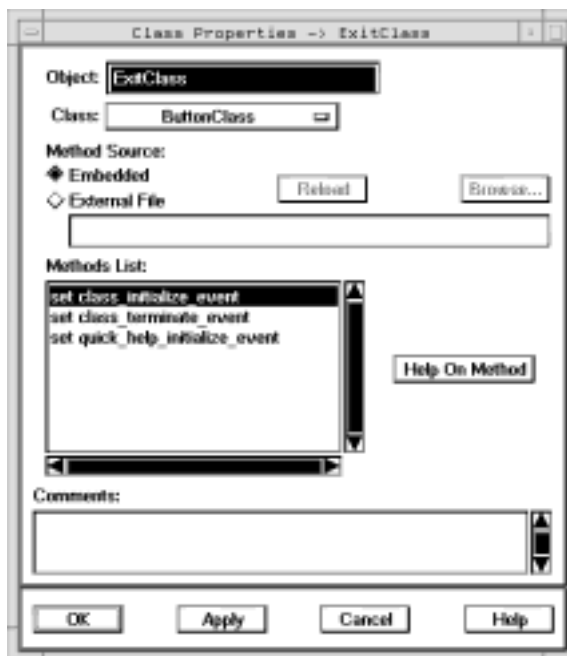
The `class_terminate_event` is reserved for future use.

For example, you can create a class called `ExitClass` that inherits attributes from the `ButtonClass`. The `ExitClass` can have a `clicked_event` to exit the application. A typical `clicked_event` for the `ExitClass` is:

```
set clicked_event
  this.application@.quit@
endset
```

To change the properties of a user-defined class:

1. Select the class in the Class Browser window.
2. Choose **Class** → **Properties**. The Class Properties dialog box appears.

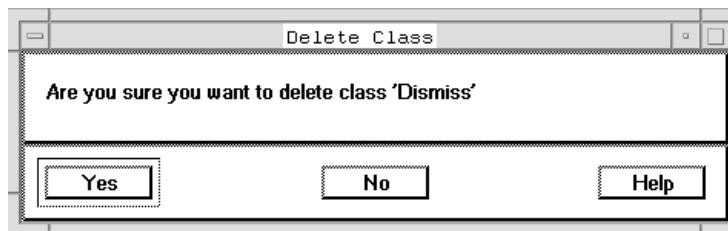


Use the dialog box to change the class name, inherited class, and the class method source file. A class method source is initially an embedded file, but you can change it to an external file and edit the source in your choice of file editor.

## Deleting Classes

A user-defined class remains as part of your application until you explicitly remove it. To remove a user-defined class:

1. Select the class in the Class Browser window.
2. Choose **Class** → **Delete**. A Delete Class dialog box appears.



3. Click **Yes** to delete the class from the application. Click **No** to cancel the deletion.

## Saving Classes

After you define a class you can save it as an external class for use in multiple applications. To save a user-defined class as an external class:

1. Select the class in the Class Browser window.
2. Choose **File** → **Save As**. A Save As dialog box appears.

Specify a name, location, and permissions for the class file. You can save the file with any file extension. However, the Builder default search when loading an external class is for files with a .dll file extension.

3. Click **Save**.

## Loading Classes

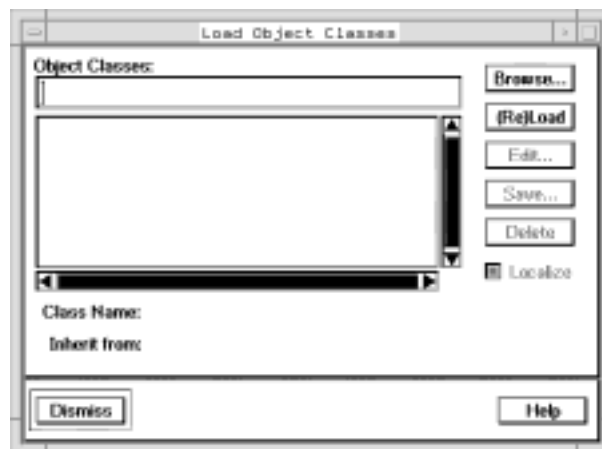
You must first load an external object class to make it available in your application. You can load an external object class as either a linked object class or a localized object class.

A linked object class references the loaded object class file. The advantage of using a linked object class is providing global updates to applications referencing the class. Changes or additions to class methods are available to the application during execution. Moving or deleting an object class file, however, causes errors in applications as the class methods are unavailable.

A localized object class keeps a copy of the external object class with the application. A localized object class does not receive changes or additions to class methods made in the external object class file. A localized object class becomes part of the application. The methods in the class are always available until the class is deleted from the application.

To load an external object class:

1. Choose **Resources** → **Load Classes** from the Browser window. The Load Object Classes dialog box appears.



2. Click **Browse**. The Load Object Class dialog box appears.



Use the dialog box options to search directories and files for the external object class. The default search is for files with a .cl? file extension. Change the search wildcard if your external object classes have a different file extension.

3. Choose the class in the Load dialog box and click on **Open**. The class appears in the Object Classes entry area.
4. Click on **(Re)Load** to load the class into the application. The class appears in the Object Classes list area.
5. Turn on the Localize option to save the class with the application, or turn off the option to keep the class linked to the file.
6. Repeat steps 2-5 to load additional object classes. Click on Dismiss when you are finished adding external object classes.

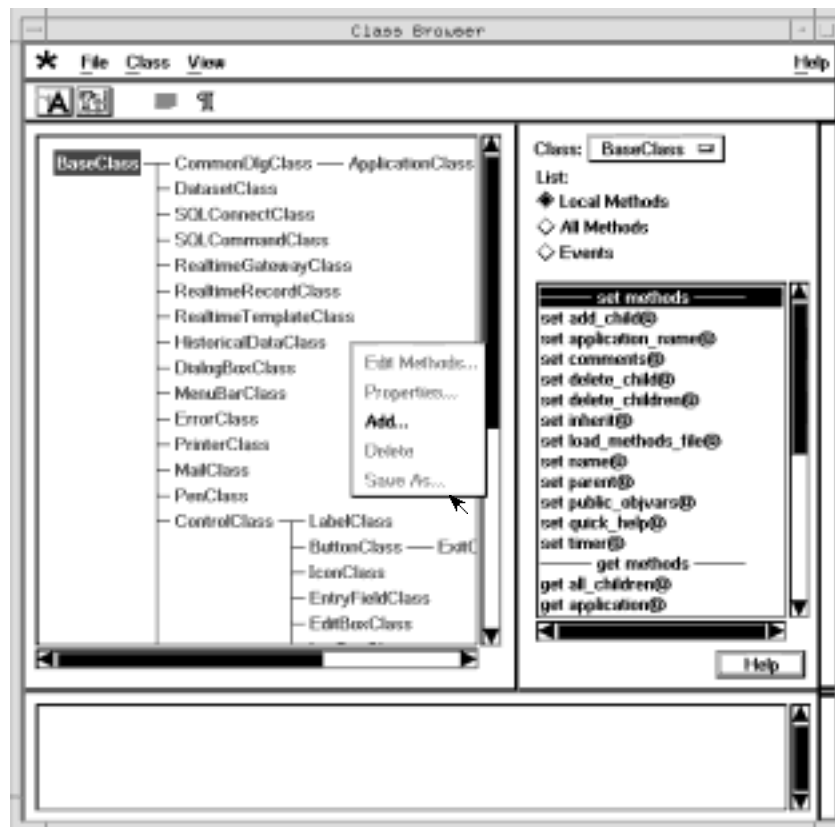
Use the Load Object Classes dialog box to delete external object classes that are no longer needed in the application. To delete an object class:

1. Choose **Resources** → **Load Classes** from the Browser window.
2. Click on the object class in the Object Classes list area.
3. Click on **Delete**.

After you add an external object class to an application you can set current or added objects to inherit attributes from the class. An added external object class appears in the Class Browser, and is available in the Connector tool or the Object Properties dialog box. For example, you can use the Connector tool or the Object Properties dialog box to change the class of a ButtonClass object to a CancelClass object. You do not have to change the object's source or methods, it automatically inherits the attributes of its newly assigned class. For example, when you click on an ExitClass object during application execution the clicked\_event for the object class is executed.

## Popup Menu

A popup menu is available in the Class Browser window to provide quick access to the class editing menu options. When the mouse pointer is in the Browser area of the Class Browser window and you press the right mouse button the popup menu appears. If you choose a menu option and release the mouse button the editing action occurs and the popup menu disappears.



## Application File Options

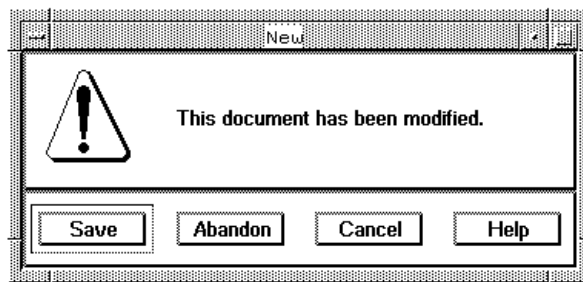
The Browser window has options under the File menu for manipulating applications. The options are:

- New
- Open
- Save
- Save As
- Send
- Print
- Revert
- Delete
- Exit

The File menu also contains the Compile and Run options. See Chapter 7, "Running and Distributing Applications", for information about using these options.

### Opening a New Application

To open a document, choose File → New. If changes are made to the existing document in the application window, the New dialog box appears. Click on **Save** to save your changes, or click on **Discard** if you do not want to save your changes.



After you save or discard the existing file, a new application is started.

The name of the top-level `ApplicationClass` object in the new application is named `Application` by default. To change the name of the application:

1. Select the `ApplicationClass` object in the Browser window.
2. Choose **Object** → **Properties**.

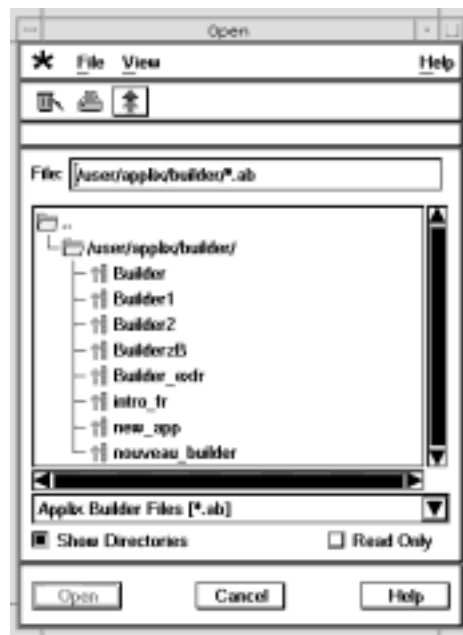
Use the object properties to change the name and other attributes of the `ApplicationClass` object. See "Changing Object Properties" later in this chapter for more information on changing object names and other properties.

## Opening an Application

Choose **File** → **Open** to display the list of applications in your current working directory.

Open an application listed in the Open dialog box using either of the following methods:

- Click on the name to select it, then click on **Open**.
- Double-click on the name to select and open the application.

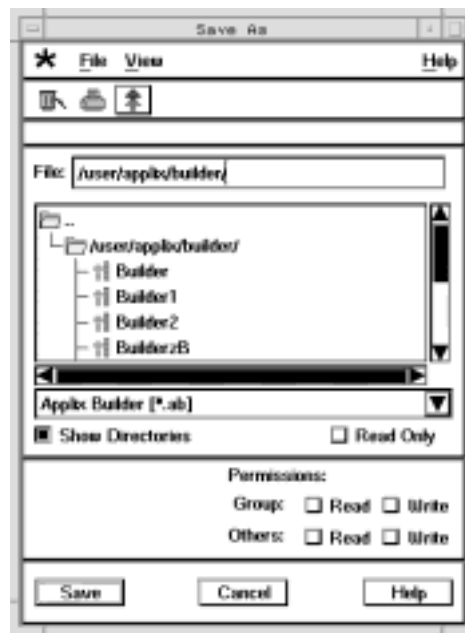


Use the File menu options to open applications located in different directories.

## Saving a New Application

To save a new application for the first time:

1. Choose **File** → **Save** or **File** → **Save As** to display the Save As dialog box.



2. Type an application name in the File entry area.

See your operating system documentation for document naming rules. An Applix Builder application is an operating system file; you can name an application just as you name a file.

Do not use the following characters when you name your document: \*, ?, <TAB>, ", [ ], ^, ~, <SPACE>, @, &, and /

The application is saved with a .ab file extension.

3. To assign read or read/write access to the application to users in your operating system group, click on the Read or Write buttons for the Group.

4. To assign read or read/write access to an application to any user, click on the Read or Write buttons for Others.

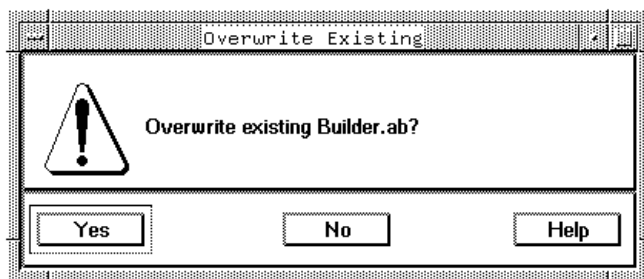
Write permission implies Read, and Others includes Group. If you select a permission that includes another level, the corresponding permissions are turned on or off automatically.

For example, if you turn on Write the corresponding Read is turned on automatically, and if you turn on Others the corresponding Group permission is turned on. If you turn off Read permission for the Group, all other permissions are turned off.

5. Click on **Save**.

The application is saved. It remains open in the Browser window and the name you assign it is displayed in the title bar.

If an application with the name you type exists in the current directory, Applix Builder displays the Overwrite Existing dialog box.



Click on **Yes** to overwrite the existing application.

Click on **No** to cancel the save operation and redisplay the Save As dialog box.

## Saving an Application under a Different Name

To keep several versions of an application, you can save a new version under a different name.

To save an application under a different name:

1. Choose **File** → **Save As** to display the Save As dialog box.
2. Type a new name in the File Name entry area.
3. Click on **Save**.

Applix Builder saves the application and any changes you've made since the last save operation to a new application with the name you assign it. The new application is displayed in the window.

Two application now exist: an old version with the old name that reflects the state of the application as of the last save operation, and a new version with the new name that contains your most recent changes.

## Sending an Application

You can use Applix Mail to send an Applix Builder application via e-mail. To send an application by e-mail:

1. Choose **File** → **Send**. The Send Document dialog box appears.
2. Choose the message recipients, set the subject, and type a message using the dialog box options.
3. Click **OK**.

The application is sent to the recipients.

## Printing Object Sources

You can print object method sources from the Browser in several ways. The File → Print submenu contains three options:

- All Sources
- Subtree Sources
- Object Sources

Choose All Sources to print the object method source of all objects in the application. If an object method source contains method, event, or macro code the source is printed to the destination printer.

Choose Subtree Sources to print the object method source for the selected object, and all of the selected object's descendants. You can use this option to print only part of an application's sources, such as printing the sources for a single dialog box.

Choose Object Sources to print the object method source for the selected object. This allows you to print a single object method source without opening the source in a Source Editor window.

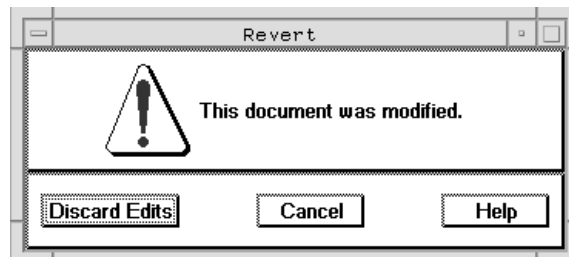
When you choose any of the Print menu options the Print dialog box appears. Use the dialog box options to select the destination printer and other print options.

## Reverting an Application

If you decide not to keep any of the changes made to an application since you last saved it, you can undo them all in one step by reverting the application to its state after the last save operation you performed. Keep in mind that once you revert an application, you cannot retrieve your edits.

To revert an application:

1. Choose **File** → **Revert** to display the Revert dialog box.

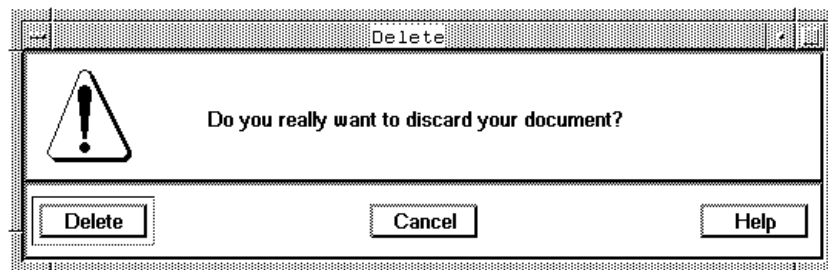


2. Click **Discard Edits** to revert the application to its previous state.
- You can also revert an application when you exit from the application.

## Deleting an Application

To delete an open application:

1. Choose **File** → **Delete** to display the Delete dialog box.



2. Click **Delete**.

The application is removed from the operating system and its window is closed.

## Exiting an Application

To close an application and its window:

1. Choose **File** → **Exit** to display the Exit dialog box.



2. Click **Save** to save your changes, or click **Discard** if you do not want to save your changes.

If you click **Save** and the application is new, the **Save As** dialog box is displayed. Complete it as described in “Saving a New Application” earlier in the chapter.

The application and its window are closed.

## Adding and Deleting Objects

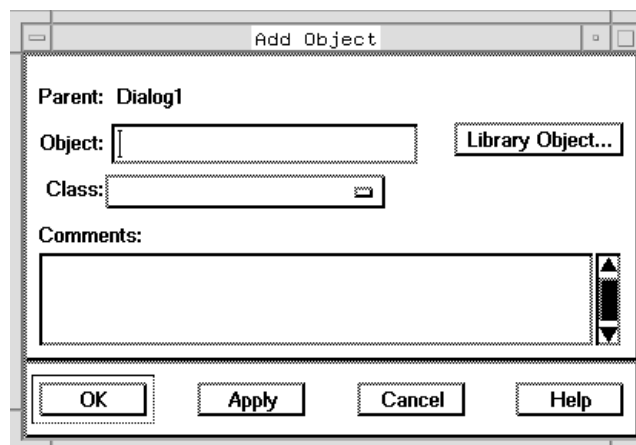
Use the Browser window to add objects to an application. For example, you can add records directly to a `RealtimeGatewayClass` object in the Browser window, instead of using the multi-step procedure in the Source tool. You can also use the Browser window to delete objects from an application, such as a dialog box or data source not used by an application.

Use the Browser window to add or delete objects only after you are familiar with creating and modifying objects with the Builder tools. You need to define the methods and properties of an added object for it to behave correctly in an application. For example, after you add a `ButtonClass` object to a dialog box with the Browser window, you must edit the object's source to set the position in the dialog box, title, color, and events.

### Adding an Object

Add objects to an application as a child of an existing object. To add an object:

1. Select an object in the Browser window.
2. Choose **Object** → **Add**. The Add Object dialog box appears.



3. Type the name of the object in the Object entry area.
4. Choose the class to inherit attributes from with the Class option.
5. Type comments about the object, such as creation or purpose, in the Comments entry area.
6. Click **OK**. The object is added as a child of the selected object.
7. Choose **Object** → **Edit Source** to edit the new object's methods and events.

See "Using a Library Object" in Chapter 8, "Advanced Topics", for information about adding a library object to an application.

### Adding an Object Dynamically

You can add an object to your application dynamically using the macro `OBJECT_CREATE@`. `OBJECT_CREATE@` works slightly differently, depending on the type of object you are trying to add.

If you are adding an object created from one of the base Builder classes, such as a push button or a words class, call `OBJECT_CREATE@` with the following arguments:

- The name of the class from which you are creating the object.
- The name of the object you are creating.

If you are adding an object created from a class you defined yourself, call `OBJECT_CREATE@` with the following arguments:

- A class object reference, as returned by the `class_library@` method of the Builder Base class.
- The name of the object you are creating.

The `class_library@` method returns a class object reference, which is then passed to the `OBJECT_CREATE@` macro to instantiate a new object within your application. The following example shows how to create an object from a user-defined class:

```
set initialize_event
  var object x, foo
  x = this.application@.class_library@("myclass")

  foo =object_create@(x,"myclass")
  foo.jump_event()
endset
```

## Deleting an Object

To delete an object with the Browser window:

1. Select an object in the Browser window.
2. Choose **Object** → **Delete**. The Delete Object dialog box appears.



3. Click **Yes** to delete the object. Click **No** or **Cancel** to exit the dialog box without deleting the object.

The object is deleted from the application, along with any children of the object.

## Cutting, Copying, and Pasting Objects

You can cut, copy, and paste objects in an application or between multiple applications. Cut or copied objects retain all attributes and edits in their source. The children of cut or copied objects are copied with the object, all children retain their hierarchical relationship with the parent object.

To cut an object:

1. Select the object in the Browser window.
2. Choose **Object** → **Cut**.

The object is cut, with all its children, from the application and placed in the clipboard. Only one object is on the clipboard at one time. Cutting or copying an object replaces the contents of the clipboard with the latest object. Previously cut or copied objects are discarded.

To copy an object:

1. Select the object in the Browser window.
2. Choose **Object** → **Copy**.

A copy of the selected object is placed in the clipboard.

To paste an object:

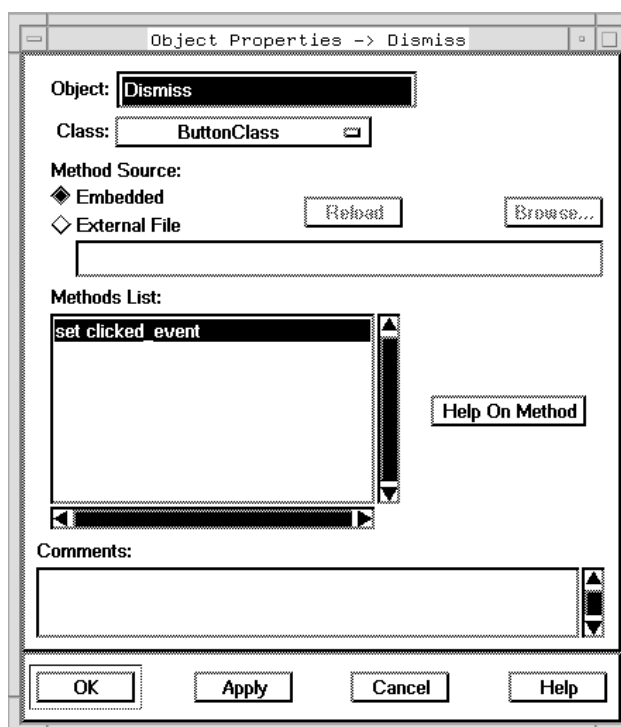
1. Cut or copy an object in the Browser window.
2. Select an object in the Browser window that will be the parent of the pasted object.
3. Choose **Object** → **Paste**.

The object is pasted, with all of its children, into the application as a child of the selected object.

## **Changing Object Properties**

Application object properties determine the actions and attributes of an object. Use the Object Properties dialog box to change the name, class, or source of an object.

Choose Object → Properties to open the Object Properties dialog box.



Change the object name or the class it inherits properties from with the appropriate dialog box options.

## Setting the Object Method Source

The object method source is saved, by default, as an embedded part of an application. You can also create an object source as an external file that you modify with your system editor and load in as the object source. External files loaded as an object source must be compilable by Applix Builder. The ELF directive `@@@ OBJECTS` must be the first non-commented line in the file to activate Builder mode. The ELF directive `@@@ OBJECTS` indicates the source contains Builder code.

See Chapter 2 in the *Applix Builder Programmer's Reference*, "Object--Oriented Concepts", for more information on Builder programming.

To load the object source from an external file:

1. Select an object in the Browser window.
2. Choose **External File** with the Method Source option.
3. Type the full path name of the file in the entry area, or click on Browse.

An Open dialog box appears when you click on Browse. Use the Open dialog box options to choose a directory and file to open. The default search is for files with a .am file extension. Change the search criteria to open files with different, or with no, file extension.

4. Click **Reload**.

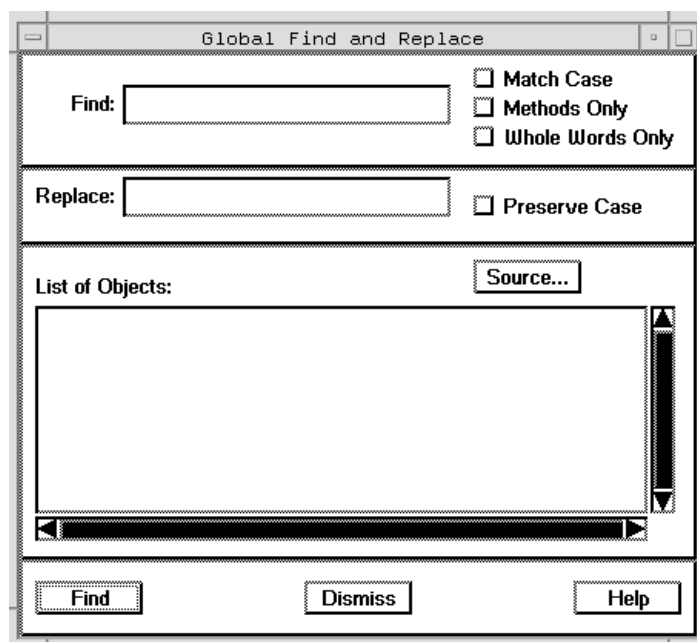
The external file is loaded as the object source. Choose Object → Edit Source to make additional edits with the Edit Source tool, or make edits to the external file with your system editor and reload the file as the object source.

## Global Find and Replace

The Browser window provides a global search of the application for words in object method sources. You can use the global find and replace to update multiple objects, instead of searching object by object for a specific term. The search begins from the selected object down

through all of the selected object's descendants. To use the global find and replace feature:

1. Choose **Tools** → **Global Find & Replace**. The Global Find and Replace dialog box appears.

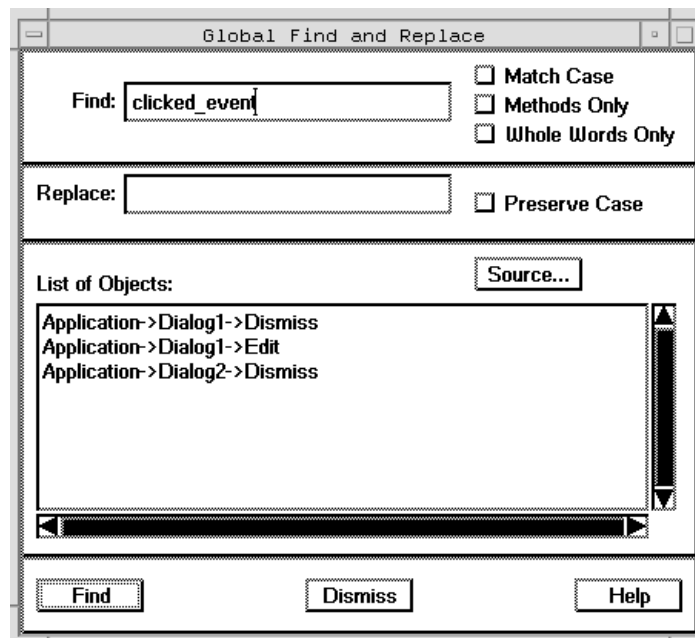


2. Type the name of the string to find in the Find entry area.
3. Choose the find options.

Toggle on Match Case to search for strings matching the case of the search string. Toggle on Methods Only to search for method names containing the search string. Toggle off Methods Only to search the entire object method source for a matching string. Toggle on Whole Words Only to search for the string as a whole word.

4. Type the replacement string in the replace entry area.
5. Click **Find**.

If a match is found, the the full hierarchy of each object matching the search criteria appears in the List of Objects list area.



Double-click on an object name to launch the local find and replace for the object method source. The object method source is opened in a Source Editor window, and the Find & Replace dialog box appears with the find and replace criteria.



---

# 3 Using the Source Tool

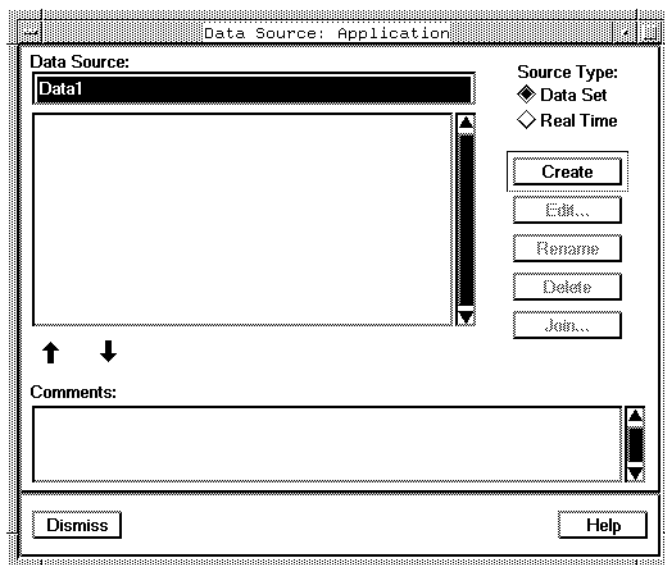
---

A useful Applix Builder application provides easily accessible, accurate information. The first step in creating a Builder application is establishing a data source. Use the Source tool to create a data set or Real Time data source. This chapter covers the following topics:

- Using the Data Source dialog box
- Creating a data set
- Creating a Real Time Data Source

## Using the Data Source Dialog Box

The starting point of a data-centric Applix Builder application is the data source. Use the Data Source dialog box to manipulate data set and Real Time data source sources. This section reviews the Data Source dialog box features.



With the Data Source dialog box you can:

- create a data set or Real Time data source
- edit a data source
- rename a data source

- delete a data source
- join two or more data sets

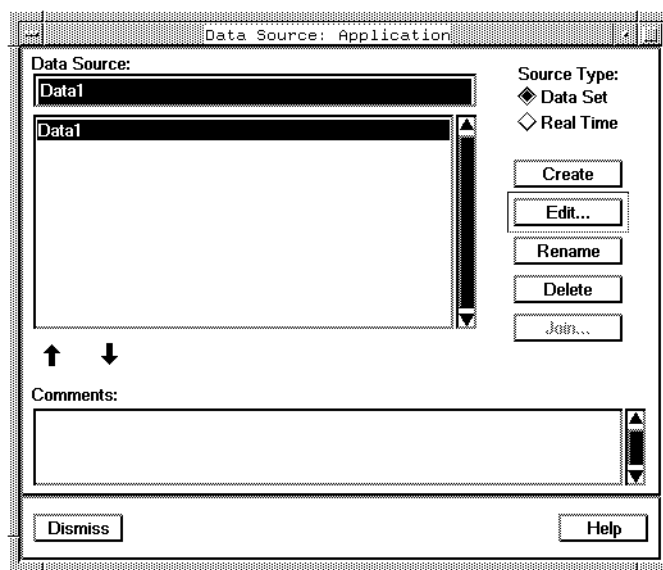
### **Creating a Data Source**



Source icon

To create a data source with the Data Source dialog box:

1. Click on the Source icon in the Browser window. The Data Source dialog box appears.
2. Type the data source name in the Data Source entry box.
3. Click on **Create**. The data source name appears in the Data Source list area.
4. Choose the source type with the Source Type option: Data Set or Real Time.



## Editing a Data Source

You can edit an existing data source to initialize or change the data source information. To edit a data source:

1. Click on the Source icon in the Browser window.  
The Data Source dialog box appears.
2. Choose a data source in the Data Source list area.
3. Click on **Edit**.

The associated data source window or dialog box appears. See "Creating a Data Set" later in this chapter for information on editing a

data set. See "Creating a Real Time Data Source" later in this chapter for information on editing a Real Time data source.

## Renaming a Data Source

You can rename an existing data source to reflect changes or as a summary of information in the data source. To rename a data source:

1. Click on the Source icon in the Browser window. The Data Source dialog box appears.
2. Choose a data source in the Data Source list area.
3. Type the new name in the Data Source entry box.
4. Click on **Rename**.

The data source name is changed to the new name.

## Deleting a Data Source

All data sources added with the Source tool remain part of an application until removed. To delete a data source from an application:

1. Click on the Source icon in the Browser window. The Data Source dialog box appears.
2. Choose a data source in the Data Source list area.
3. Click on **Delete**.

The data source is deleted. The data source name is removed from the Data Source list area.

## Creating a Data Set

A data set is an SQL query on one or more database tables created with a Source data window. When you create complex database queries with the Source data window, Applix Builder automatically generates the SQL source code.

To create a data set with the Data Source dialog box:

1. Click on the Source icon in the Browser window.

The Data Source dialog box appears.

2. Type the data set name in the Data Source entry box.
3. Click on **Create**.

The data set name appears in the Data Source list area.

4. Choose **Data Set** from the Source Type option.
5. Click on **Edit**.

The Source data window for the data set appears.

## Selecting a Database

You can select a database to connect to with an open Source data window. To connect to a database you require the following information from your database system administrator:

- Database vendor
- Database name

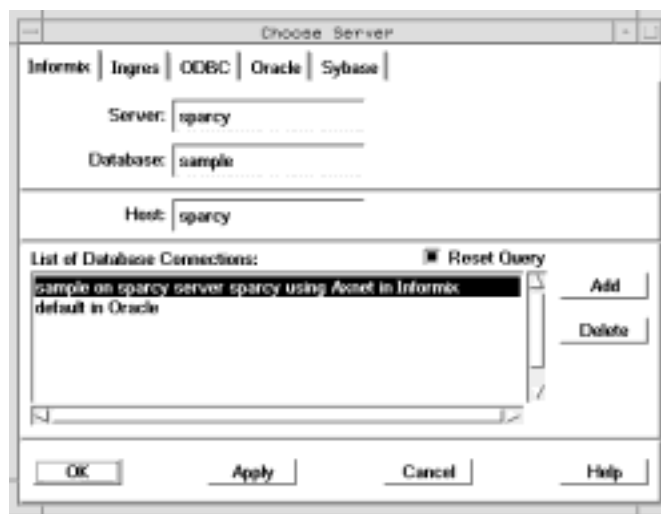
- Name of computer where the database is installed

You may also require additional information, depending on the database vendor-type, such as:

- User name
- Password
- Server
- Vendor routing

Use the Choose Server dialog box to connect to the available databases in your system. Choose Query → Choose Server to bring up the Choose Server dialog box. A database must be in your list of available databases before you can open it. To add a database to your list of available databases:

1. Choose **Query** → **Choose Server** in the data window. The Choose Server dialog box appears.



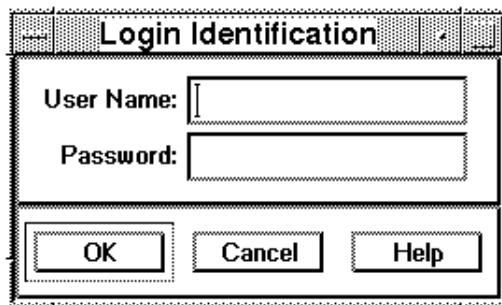
2. Choose the database vendor type from one of the vendor tabs.
3. Click on **Add**.
4. Type the name of the computer on which the database server is running in the Host entry box.
5. Type the database name in the Database entry box.
6. Depending on the vendor-type, you may have to enter information in the other entry boxes. Information is not required for areas with grayed titles.
7. Click on **OK** to add the database and connect to the database.

You can add as many databases to the database list as you require. After you add a database to the list of available databases, you can open it. To open a database:

1. Choose **Query** → **Choose Server**.

2. Click on the database name from the list of available databases.

To connect to some vendor databases you must enter a user name and password. Choose **Tools** → **Login Identification** to enter your database user name and password.



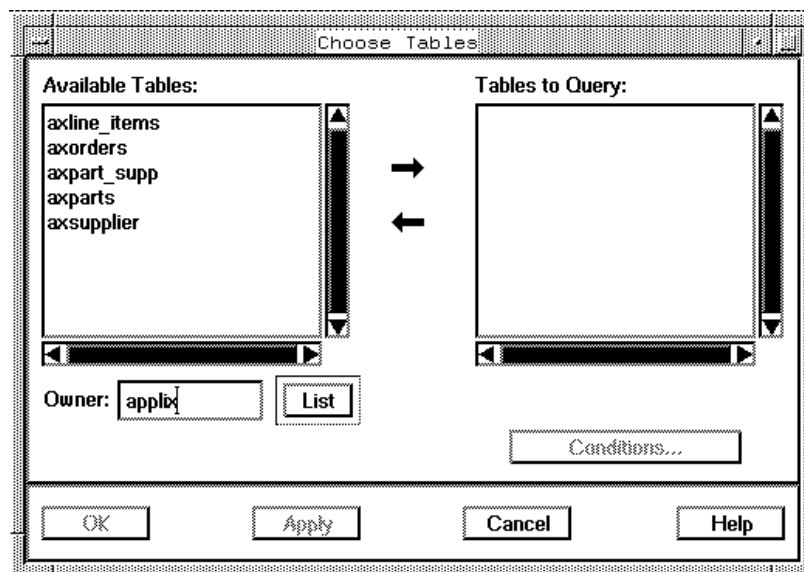
The image shows a dialog box titled "Login Identification". It contains two text input fields: "User Name:" and "Password:". Below the input fields are three buttons: "OK", "Cancel", and "Help". The dialog box has a standard Windows-style border with a title bar and window control buttons.

When a link is established to the database the Ready prompt appears in the status line at the bottom left of the Source data window. If the database information is incorrect, or if you do not have permissions to the database, an error message appears.

## Selecting a Table

Once you establish a connection to a database you must choose a table to view. After you select a table, you can refine your query to look for specific information. To choose a table:

1. Choose **Query** → **Choose Tables**.



2. Scroll through the Available Tables list area until you find the table you want to view.
3. Double-click on the table name to move the table to the Tables to Query list area.

The Querying database prompt appears in the status line of the Source data window while Applix Builder is querying the database.

All the tables in a database have an owner associated with them. For example, in the Available Tables list area, the item applix.axorders refers to the axorders table owned by applix. You can narrow your choices of tables to specific owners with the Owner entry box and the List button. To choose tables with a specific owner:

1. Double-click in the **Owner** entry box.

2. Type the owner name in the entry box.
3. Click on **List**.

The tables for the owner appear in the Tables list area. To choose all tables, double-click in the Owner entry box and press SPACEBAR, then press List.

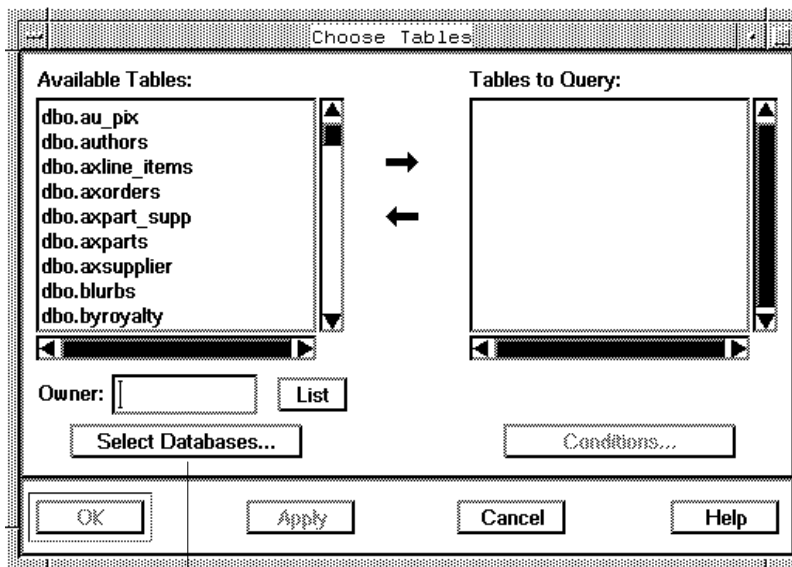
When Applix Builder is finished querying the database, Data retrieves a set of rows from the table. Beginning with the first row, retrieved rows are displayed in the Source data window. Click on the Page Down button or the down arrow in the vertical scroll bar to retrieve more table rows. When all available rows are retrieved, the Page Down button disappears, the vertical scroll bar expands, and the total number of rows retrieved appears in the Total status area.

supplier_num	lname	fname	phone	ext	company
000001	Crighton	Bill	(508)555-6881	962	Nuts R Us
000002	Rush	Jack	(508)555-8365	2387	Westboro Abrasive
000003	Teagan	Donald	(617)555-2536	39	New England Bolts
000004	Lentil	Jay	(203)555-9286		West End Fabricat
000005	Maroon	Kevin	(401)555-8952	766	Monarch Manufac
000006	Miller	Dennis	(508)555-0004	11354	Sharpe Tools
000007	Bodean	Kathy	(508)555-9898		C & H Tool and Die
000008	Deane	Frank	(508)555-1175		Drywall Fasteners
000009	Smith	Elizabeth	(508)555-3400	231	Quality Inc.
000010	Hunt	Laura	(401)555-8636		Acme, Inc.
000011	Smith	William	(508)555-6521		New England Manu
000012	Johnson	Benjamin	(508)555-1000	929	Newton Nails

Total status area

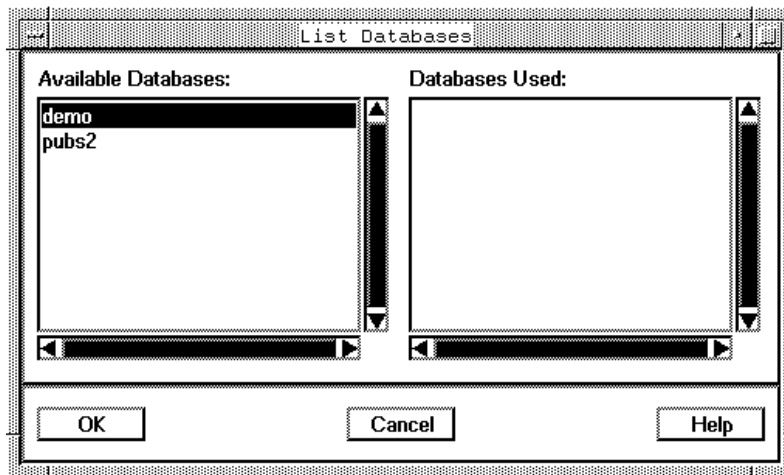
## Connecting to a Sybase Table

If you connect to a Sybase database, you can choose tables from more than one database at one time. A Select Databases button appears in the Choose Tables dialog box.



Select Databases button

When you click on Select Databases, the List Databases dialog box appears. A list of available Sybase databases appears in the dialog box.



You can use tables from more than one database at the same time. You can perform cross-database table joins in the Source data window with Sybase databases. The cross-database table joins do not appear different from a standard table join. See "Creating a Data Set Join" later in this section for information on creating logical database joins.

## Saving a Query

After you establish a database connection and choose a table you can save a query as part of the Applix Builder application. To save a query, choose File → Save from the Source data window.

After saving a query, choose File → Exit from the Source data window. After exiting the Source data window, click Dismiss to dismiss the Data Source dialog box.

## Starting a New Query

You can establish a new query in the Source data window while maintaining your connection to the currently selected database. You can choose **File → New** to clear the current database query. The connection to the database exists after choosing **File → New**. You only need to choose a new table for your query. To establish a new query:

1. Choose **File → New**.
2. Choose **Query → Choose Tables** to choose another table in the current database.

The new query is established, and the first set of rows in the new table appear in the Source data window. If you want to establish a new query with a different database, refer to "Selecting a Database" earlier in this section.

## Auto-Query and Query

You can turn on **Query → Auto-Query** to automatically query the database when you change and apply database querying settings. Choose **★ → Data Preferences** to select the Auto-Query default setting. You can turn off **Query → Auto-Query** to have the database query only when you choose **Query → Query**.

For performance reasons, you may want to turn **Query → Auto-Query** off. When you are changing the sort and query conditions, you may want to make all your changes before you query the database. Each query must search through all records of the database, and the larger the database, the longer the time required for the search. Every change in the sort and query conditions creates a new query statement. With **Query → Auto-Query** on, every time you change a condition, the database is queried.

You can use Query → Query to update your database query when Query → Auto-Query is turned off. If the information in the database has changed since your last query, you can choose Query → Query to get the most recent information. You can also click on the Query icon in the *ExpressLine*.



Query icon

The "Querying database" prompt appears in the status line when you choose Query → Query. When the query is completed the prompt disappears.

## Setting Query Conditions

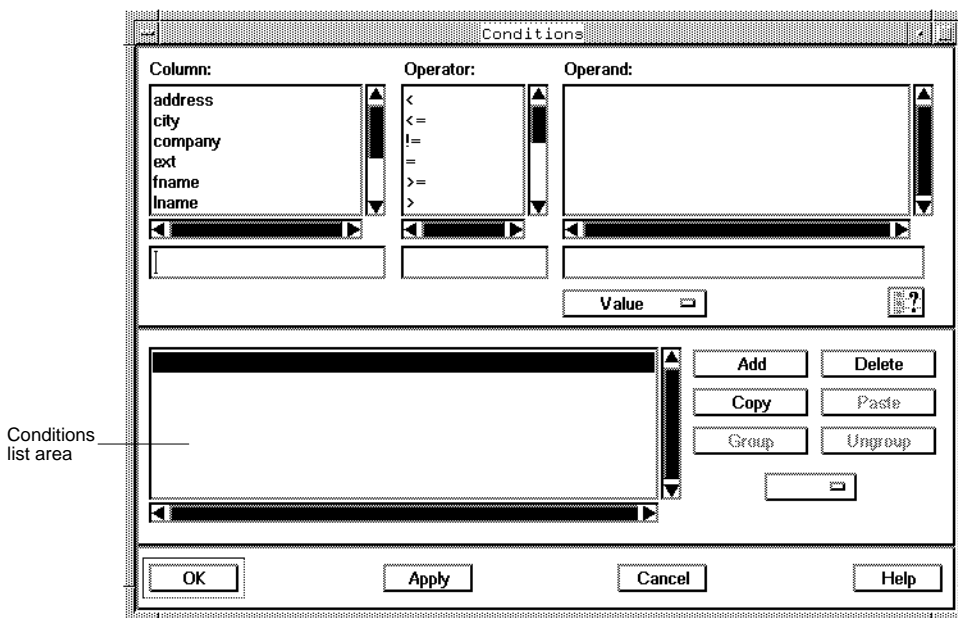
You can set conditions for your query to find specific information in the database. You can choose different sets of conditions that yield the same result. The most common query is a query for a specific value, but you can also query for columns or a subselect condition.

You can use the following to set conditions for a value query:

- Comparison operators
- In
- Between
- Like
- Is Null

To set a query condition:

1. Choose **Query → Conditions**.



2. Click on a column in the Column list area.

At this point you can choose one of the comparison operators for the query from the Operator list area. The comparison operator you choose appears in the Operator entry box.

You can type values in the Column, Operator, and Operand entry boxes. To type a value in the Column, Operator, and Operand entry boxes:

1. Click in the entry box.
2. Type the value in the entry box.
3. Press **TAB**.

After you press TAB the cursor moves to the next entry box and the value appears as part of the condition in the Conditions list area.

You can choose comparison values from the Operand list area. When you click on a value in the Operand list area, it appears in the Operand entry box. You can also type a value in the Operand entry box.



Value  
Query  
icon

The values in the Operand list area are for the first set of retrieved rows. To get the next set of values for the selected column, click on the Value Query icon. Each time you press the Value Query icon, the database is queried, and Applix Builder retrieves values from the next set of 200 records until it reaches the end of the table.

## Comparison Operators

The comparison operators are used to find column values greater than, less than, or equal to a given value. The comparison operators include:

=	Column equals value
!=	Column is not equal to value
>	Column greater than value
>=	Column greater than or equal to value
<	Column less than value
<=	Column less than or equal to value

To view rows with supplier\_num less than 000009:

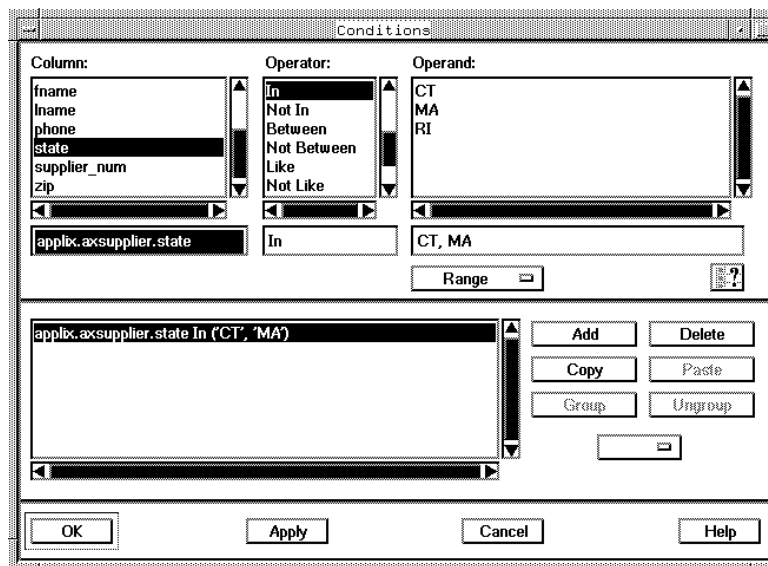
1. Choose **Query** → **Conditions**.
2. Click on **supplier\_num** in the Column list area.
3. Click on **<** in the Operator list area.
4. Click on **000009** in the Operand list area.

The query retrieves the rows meeting the query condition, the rows containing a supplier\_num less than 000009.

## In

For the In comparison operator, the query searches for rows where the column's value is in a specified list of values. You place a list of values in the Operand entry box, each value is separated by a comma. When you choose the In comparison operator, the query type changes from Value to Range. Similarly, choosing a Range query automatically sets the comparison operator to In. See "Other Query Conditions" later in this chapter for more information regarding the other query condition options.

For example, in the axsupplier table, you can choose the state column and set the query to return all rows where the state column contains CT or MA. You need at least one value in the Operand entry box for an In query, but you can have more. This example returns twelve rows from axsupplier table that meet the In condition.



Use the Not In comparison operator to search for rows where the column's value is not in a specified list of values.

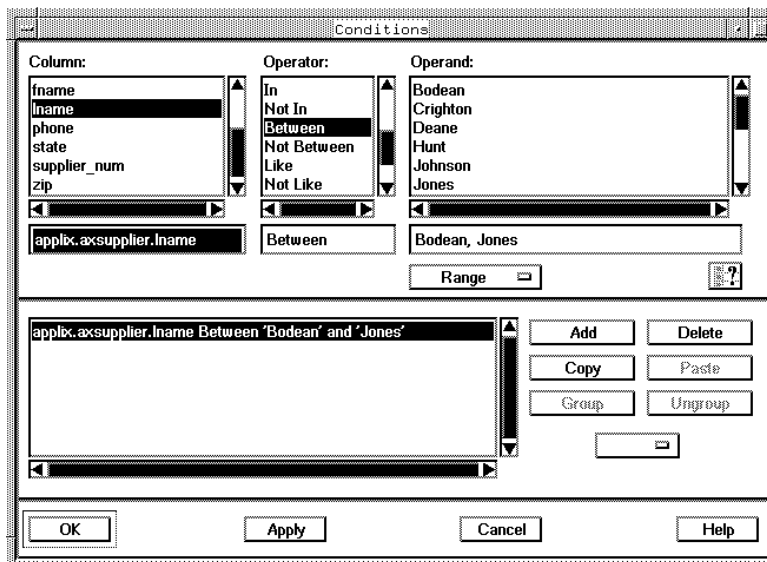
To create an In query condition:

1. Choose **Query** → **Conditions**.
2. Click on a column in the Column list area.
3. Click on **In** in the Operator list area.
4. Click on values in the Operand list area, or type values in the Operand entry box. If you type values in the entry box, separate the values with commas.

## Between

For the **Between** comparison operator, the query searches for rows containing values in the specified column that fall between two values, including the two limit values. When you choose the **Between** comparison operator, the query type changes from **Value** to **Range**.

For example, in the `axsupplier` table, you can choose the `lname` column and set `Bodean` and `Jones` as the values for the **Between** comparison. The query returns all rows with a value in the `lname` column between `Bodean` and `Jones`, including `Bodean` and `Jones`. You need two values in the **Operand** entry box for a **Between** query. Each value in the **Operand** entry box is separated by a comma. This example query returns six rows from the `axsupplier` table that meet the **Between** condition.

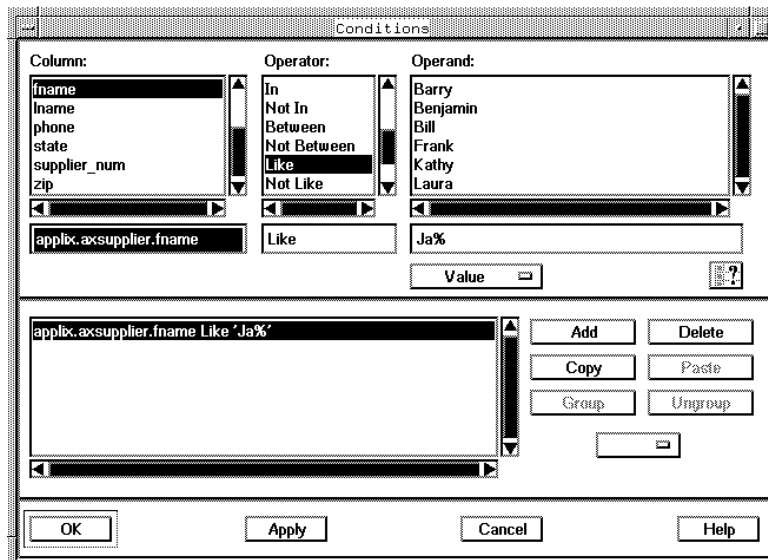


Use the Not Between comparison operator to search for rows where the column's value is not between two specified values.

## Like

For the Like comparison operator, the query searches for rows containing strings in the specified column that match a partial description of the string. The specified value is a substring of the column's text. Use wildcard characters to match the missing characters. A % will match any number of characters, an \_ matches only one character. Like searches for the exact case of the character string. For example, Car% and car% are different strings, and return different queries.

Use the Not Like comparison operator to search for rows where the column's value is not a substring of the column's text.



For example, in the `axsupplier` table, you can choose the `fname` column and search for first names beginning with `Ja`. To initiate a Like query:

1. Choose **Query** → **Conditions**.
2. Click on **fname** in the Column list area.
3. Click on **Like** from the Operator list area.
4. Double-click in the Operand entry box, and type the string **Ja** with a wildcard character.

If you type `Ja%` in the Operand entry box and apply the condition, the query returns all rows with names in the `fname` column that begin with `Ja` and end with any number of characters. This example query returns two rows from the `axsupplier` table that meet the Like condition `Ja%`.

If you type `Ja_` in the Operand entry box and apply the condition, the query returns all rows with names in the `fname` column that begin with `Ja` and end with one character. This example query returns one row from the `axsupplier` table that meets the Like condition `Ja_`.

## Is Null

For the Is Null comparison operator, the query searches for rows containing Null, or no value, in the specified column. This does *not* search for the value 0 or an empty string. Null is an empty value in a database record.

Use the Is Not Null comparison operator to search for rows where the column value is not Null.

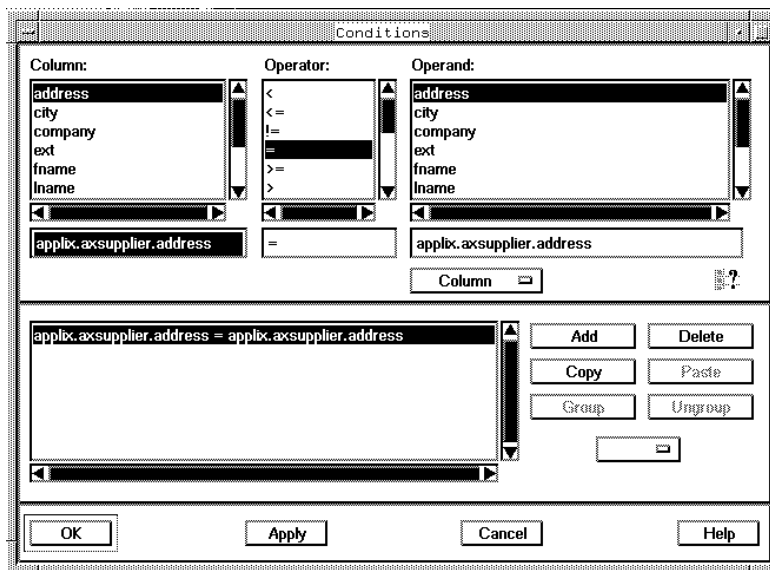
## **Other Query Conditions**

Value is the default query condition. In addition to the value query conditions mentioned earlier, there are three other types of query conditions available:

- Column
- Range
- Subselect

## **Column**

You can use the comparison operators to compare columns from the same table. If you choose more than one table for your query with Query → Choose Tables you can compare columns from different tables.



To add a column query condition:

1. Choose **Query** → **Conditions**.
2. Choose **Column** from the Condition Type option.
3. Choose a column from the Column list area.
4. Choose a comparison operator.
5. Choose a comparison column from the Operand list area.
6. Click on **OK** or **Apply** to query for the Column condition.

The comparison operators allow you to make an unequal column comparison.

## Range

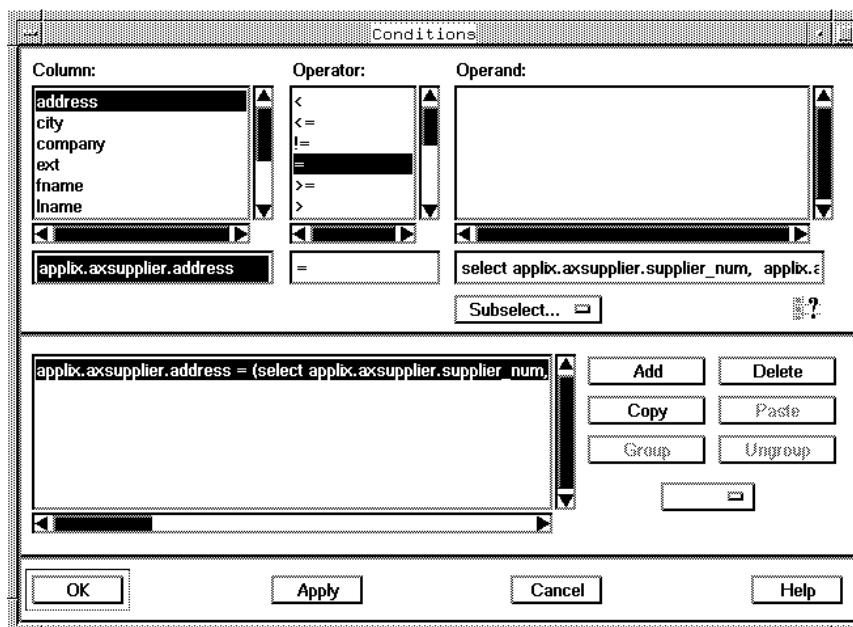
You can use the Range query condition to compare a column against a range of values. Choosing a Range query automatically sets the comparison operator to In. You can also perform a Range query with the Between comparison operator. See "Setting Query Conditions" earlier in this chapter for information on using the In and Between comparison operators.

## Subselect

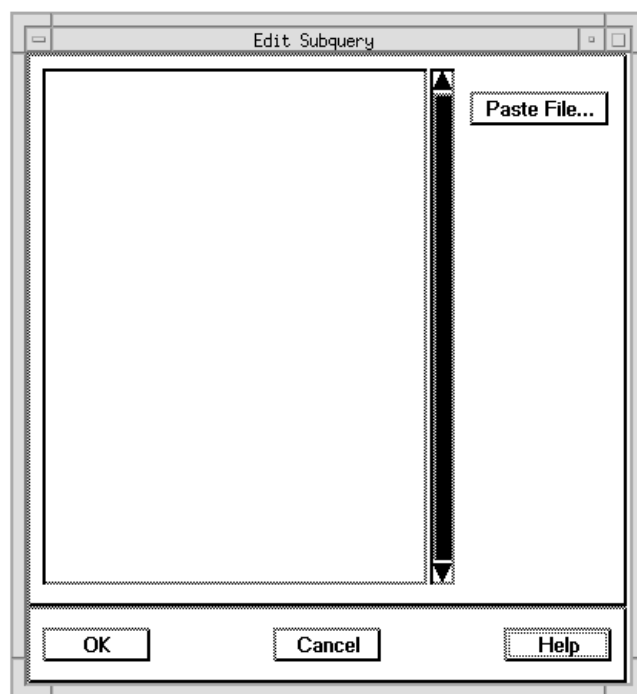
You can copy the source code of a query file and use it as a query condition. Query files include Applix Data files with an .aq extension, or SQL files with a .sql extension.

To add a subselect query condition:

1. Choose **Query** → **Conditions**. The Conditions dialog box appears.



2. Choose **Subselect** from the Condition Type option. The Edit Subquery dialog appears.



3. Click **Paste File**. The Subselect dialog box appears.
4. Choose the file from the Subselect dialog box. If the file is not in the current directory, use the directory display to choose files in another directory.
5. Click on **Open**. The contents of the selected file appear in the Edit Subquery list box. You can edit material by clicking in the list box and entering new text.
6. Click **OK**.

7. Choose a column from the Column list area of the Conditions dialog box.
8. Choose a comparison operator from the Operator list area. The default comparison operator is =.

## Multiple Query Conditions

You can set multiple query conditions in the Conditions dialog box. Multiple conditions help you to further narrow your search. Click on Add Condition to add more conditions to your query. Added conditions appear above the currently selected condition in the Conditions list area.

When you add more than one condition, a logical operator appears on the lines preceding the additional conditions. By default the logical operator is AND, but you can change the operator to OR with the AND/OR option.

You can also remove conditions from the Conditions list area with the Delete Condition button. To delete a condition:

1. Click on the condition in the Conditions list area.
2. Click on **Delete Condition**.

## AND

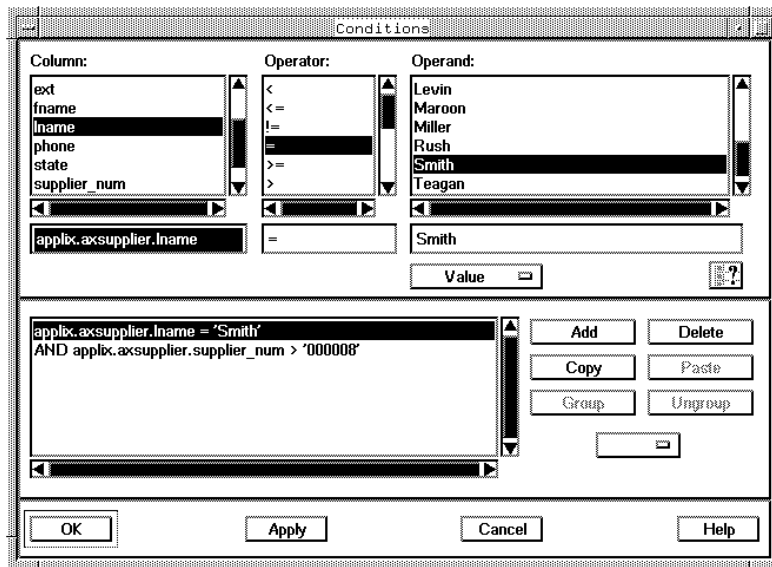
A logical AND is known as the intersection of two or more conditions. A row must meet all conditions of AND conditions, such as color = blue AND vehicle = car. Blue buses or red cars would not match these conditions. Only blue cars match the query condition.

For example, the axsupplier table in the sample database contains list of suppliers and their companies. You can find out how many

suppliers have both a supplier number greater than 8 and a last name of Smith.

To query for the multiple AND conditions, after connecting to the database and choosing the axsupplier table:

1. Choose **Query** → **Conditions**.
2. Click on **supplier\_num** in the Column list area.
3. Click on **>** in the Operator list area.
4. Click on **00008** in the Operands list area.
5. Click on **Add**.
6. Click on **Iname** in the Columns list area.
7. Click on **=** in the Operators list area.
8. Click on **Smith** in the Operands list area.



The query retrieves only two rows that have both a supplier number greater than 8 and last name of Smith.

OR

A logical OR is known as the union of two or more conditions. A row must meet at least one of the conditions, such as color =blue OR vehicle = car. Blue buses, red cars, as well as blue cars match the query condition.

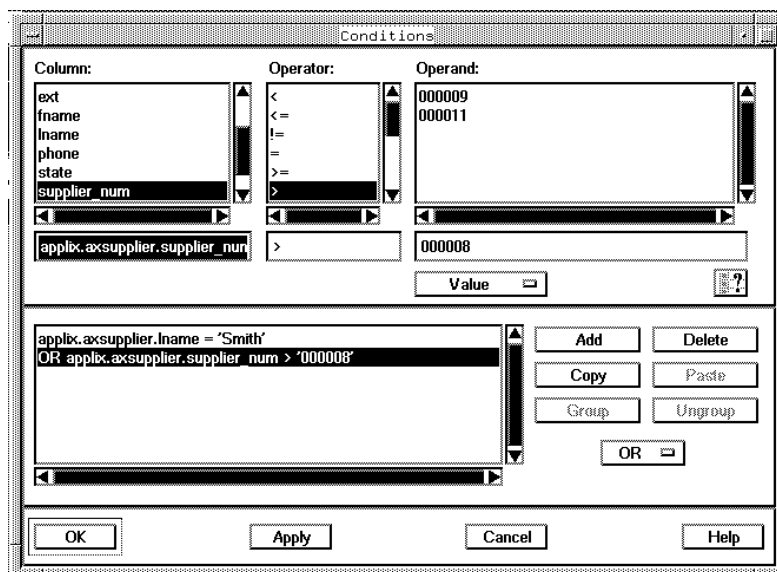
For example, the axiosupplier table in the sample database contains list of suppliers and their companies. You can find out how many

suppliers have either a supplier number greater than 8 or a last name of Smith.

To query for the multiple OR conditions, after connecting to the database and choosing the axsupplier table:

1. Choose **Query** → **Conditions**.
2. Click on **supplier\_num** in the Columns list area.
3. Click on **>** in the Operators list area.
4. Click on **00008** in the Operands list area.
5. Click on **Add**.
6. Click on **Iname** in the Columns list area.
7. Click on **=** in the Operators list area.
8. Click on **Smith** in the Operands list area.
9. Click on the second line in the Conditions list area with the AND logical operator.
10. Choose **OR** from the AND/OR option.

The logical operator changes from AND to OR.



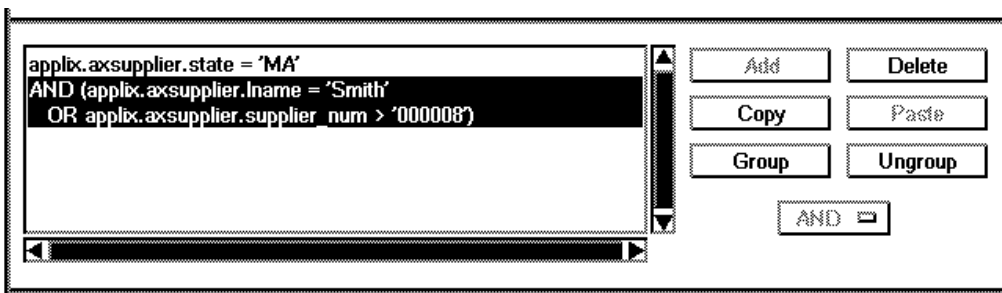
The query retrieves seven rows that either have a supplier number greater than 8 or a last name of Smith.

## Parentheses

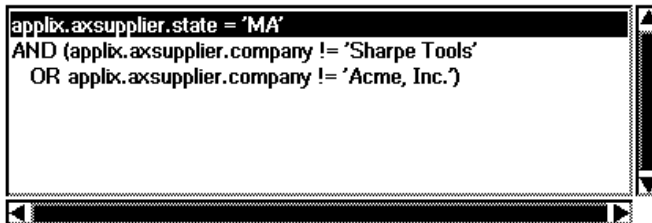
You can change the grouping of conditions with parentheses. Conditions in parentheses are applied first regardless of logical operators, before the logical preferences of the operators outside parentheses. The logical AND has precedence over the logical OR, so conditions without parentheses are grouped with AND, then OR.

To group conditions with parentheses:

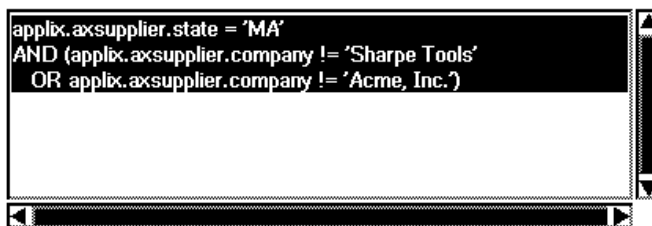
1. Click on a condition and drag up or down to choose the conditions you want to group.
2. Click on **Group**.



A parentheses appears around the conditions in the Conditions list area. If you make a selection that cannot be grouped, the selection automatically extends to the next selectable group. For example, in the following figure, the second and third conditions are grouped.



If you try to select the first and second conditions in order to group them, the selection extends to the third condition because you cannot split grouped conditions.

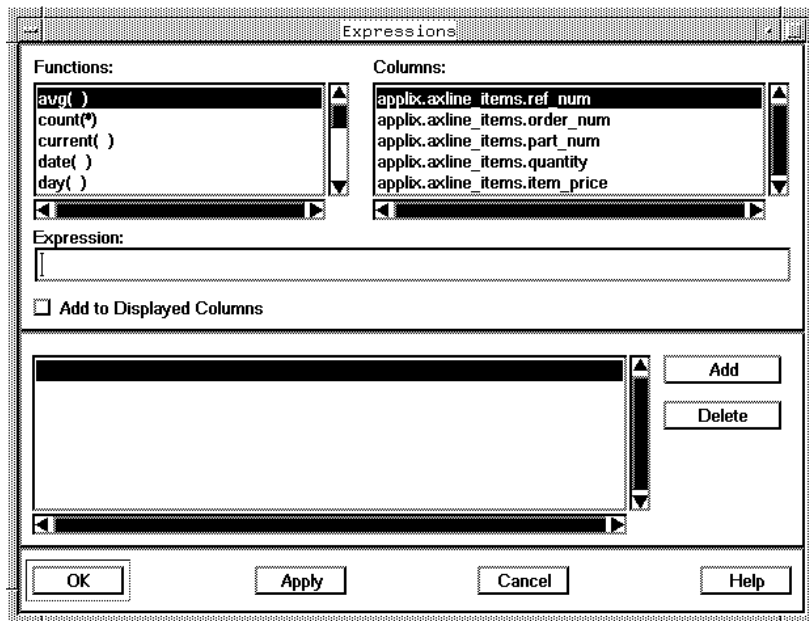


To remove parentheses from conditions:

1. Choose the conditions in the Conditions list area grouped with parentheses.
2. Click on **Ungroup**.

## Adding Columns

You can create columns with the Expressions dialog box using information from the table columns in your query and simple mathematical functions, or aggregate SQL functions, such as sum (sum), average (avg), maximum (max), and minimum (min). Although you can change column headings, as described later in this chapter, you cannot use column heading aliases in an expression. You must use the original column name.



For example, the axline\_items table in the sample database contains the columns quantity and item\_price.

If you want to get the total price for each row, you would multiply quantity by item\_price. After you connect to the database and choose a table, you can create a total price column in the following manner:

1. Choose **Query** → **Expressions**.
2. Click on **quantity** in the Columns list area. quantity appears in the Expression entry box.
3. Click in the Expression entry box after quantity.
4. Type **\***.

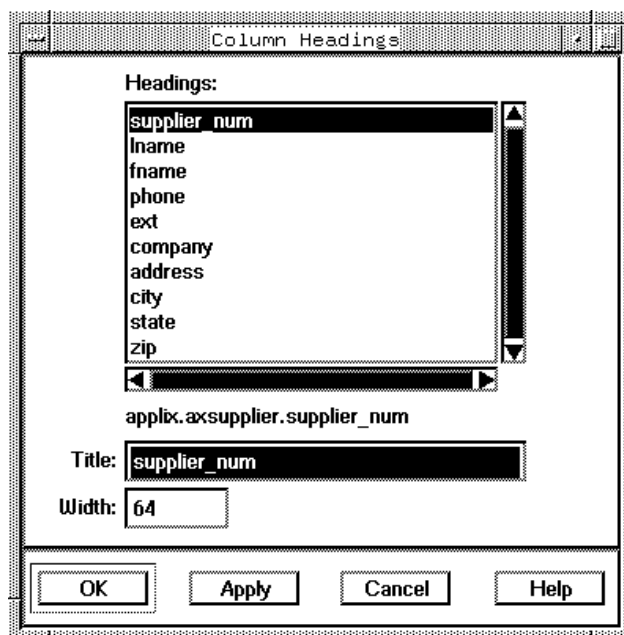
5. Click on **item\_price** in the Columns list area.
6. Click on **Add To Displayed Columns** to add the expression to the columns currently displayed in the Data window.

The column becomes part of the table query. The column heading is the actual expression.

## Column Headings

Your database query uses the table column headings and expressions you create as the column headings in your query. You can choose **View → Headings** to change the column headings, column width, and column format. You can customize the column headings and column format to make your query of the database more readable and easier to understand. To change column headings:

1. Choose **View → Column Headings**.



2. Choose a column heading from the Headings list area.
3. Type a new column heading in the Title entry box.
4. Type a new column width, in pixels, in the Width entry box.

The original column name appears above the Title entry box. Even if you change the column title, the original column name remains unchanged. You use the original column name, as described previously in this chapter, to create expressions.

You can also change the column width in the Source data window with the mouse. Move the mouse over the border between the headings to change the text cursor to  $\leftrightarrow$ . Drag the border right to expand the column, left to contract the column. All other cells shift accordingly.

## Group by and Having

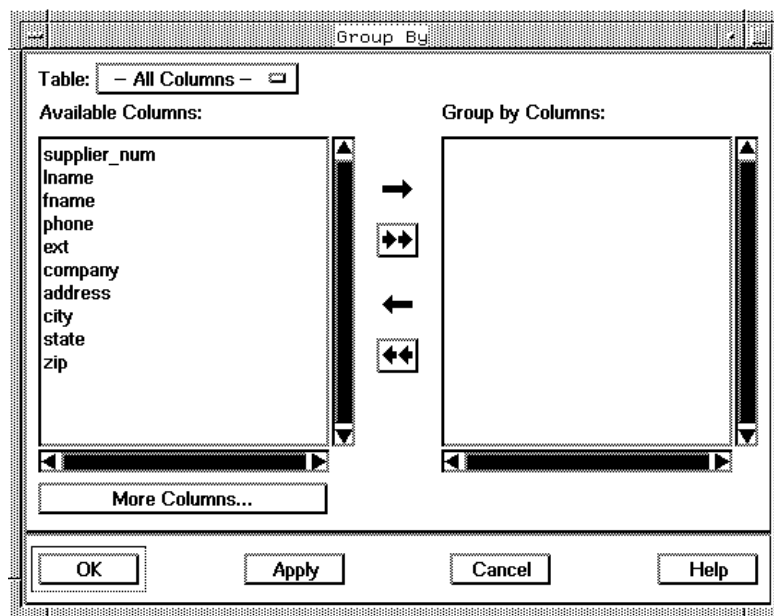
You can use the Query → Group by option to divide the information in a table into related sets. Aggregate functions were introduced in the "Adding Columns" section of Chapter 4 of the *Data* manual, "Editing Database Information". Aggregate functions create values for the distinct Group by sets. An aggregate value on a Group by set is also known as a vector aggregate. You can add a column with an aggregate function to create values and display them as part of your query.

You can group sets of rows by choosing Query → Group by. You can use Group by to assemble rows in groups sharing the same feature. After you group the rows you can set conditions and use aggregate functions on the groups. For example, you can use Group by and aggregate functions to group rows in an orders table by part numbers to determine the total number of each part ordered.

The columns that are not chosen as Group by columns are automatically removed from the query. The columns that are not chosen as Group by columns are also removed from the displayed columns. Do not include columns added to the query with Query → Expressions as Group by columns.

To group the rows in the table query:

1. Choose **Query → Group by**.

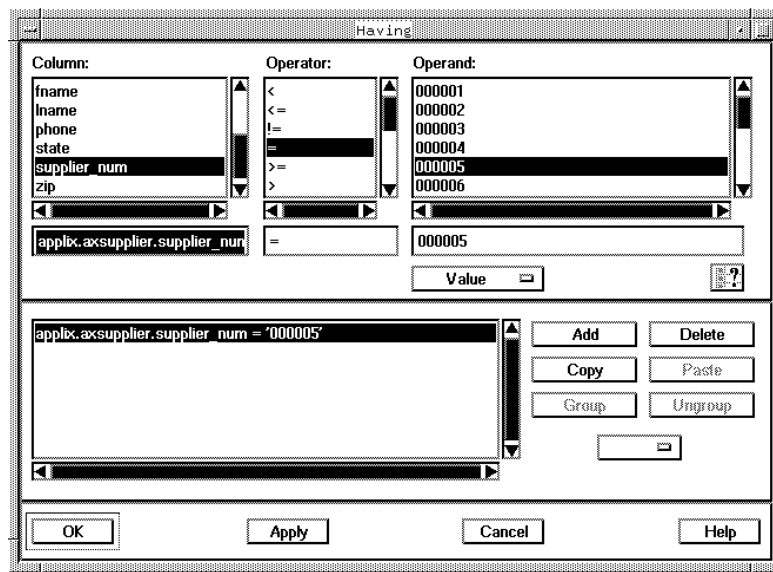


2. Move the columns you want to group, except created expressions, to the Group by Columns list area.

After the Group by database query you can choose Query → Having to refine the query for specific groups. Having applies conditions to groups of rows, while Conditions applies conditions to individual rows. Use comparison operators to define the condition. You can use NOT with the comparison operators to search for the opposite occurrence of the comparison.

After a Group by query, to set the Having condition:


1. Choose **Query → Having**.



2. Choose a column in the Column list area.
3. Choose a comparison operator in the Operator list area.
4. Choose a value from the Operand list area or type a value in the Operand entry box.

After you apply Group by and Having to the table query, you can add columns with aggregate functions to the table. A useful function for grouped rows is count (\*). The aggregate function count(\*) counts the number of selected rows. For a simple table, count(\*) would return a value of 1 for each row. For a Group by query, it returns the number of rows in each group of rows.

For example, you can use Group by and count(\*) to find the quantity of line items in each order. The axline\_items table in the sample database contains orders and line items. To set the Group by query, with Auto-Query turned on:

1. Open a Data window.
2. Choose **Query** → **Choose Tables**.
3. Choose the **axline\_items** table from the Available Tables list area.
4. Choose **Query** → **Choose Columns**.
5. Click on .
6. Double-click on **order\_num** in the Available Columns list area to move it to the Columns to Display list area.
7. Choose **Query** → **Group by**.
8. Double-click on **order\_num** in the Available Columns list area to move it to the Group by Columns list area.
9. Choose **Query** → **Expressions**.
10. Click on **count(\*)** in the Functions list area to move it to the Expression entry box.
11. Click on **Add To Displayed Columns** to add the expression to the columns currently displayed in the Data window.

The rows are grouped by **order\_num**, and the quantity of line items in each group are displayed.

## Table Join

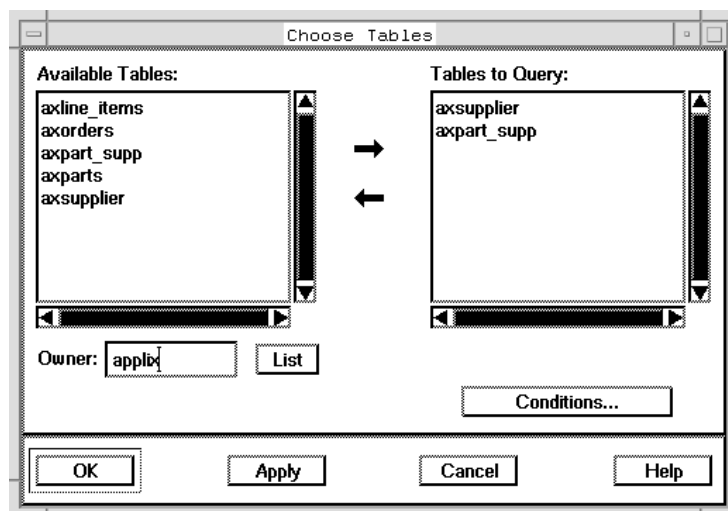
You can choose more than one table with **Query** → **Choose Tables** to form a join. The purpose of a join is to provide more information than a user can get from just one table. For example, the **axpart\_supp** table contains columns for **part\_num**, **supplier\_num**, and **supplier\_code**, but does not provide additional supplier information. You could join the **axpart\_supp** table with the **axsupplier** table, which contains the

supplier information, such as company, supplier\_num, and address. Tables are joined based on columns having the same value. In this example, the join would occur on the supplier\_num column in each table.

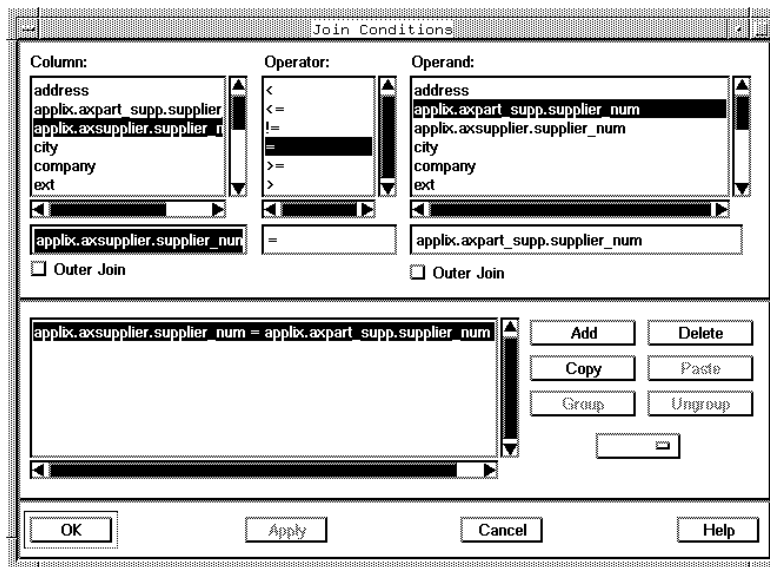
Joins created within a data set are between tables from the same database. A join on data sets allows a logical join between separate databases, even databases from different vendors, located on different machines. See "Data Set Joins" later in this section for information on creating logical joins between two or more data sets.

To create a join between two tables:

1. Choose **Query** → **Choose Tables**.
2. Double-click on the table name in the Available Tables list area. The table name moves to the Tables to Query list area.
3. Double-click on another table name in the Available Tables list area. The table name moves to the Tables to Query list area.



When you choose a second table the Conditions option becomes active. Click on Conditions to see the conditions in the Join Conditions dialog box.



Columns from the tables appear in the Column and Operand list areas. The default join columns appear in the Column and Operand entry boxes. The resulting join condition appears in the Conditions list area. Applix Builder automatically chooses the first column from each list area with the same name. If you have a Join Columns macro hook, as described in Chapter 4 of the *Applix Data* manual, the best join columns supplied by the macro are chosen. If you want to join the tables with different columns, or the columns are matched incorrectly, you can choose different columns from each list area. Turn on Outer Join to display rows with a NULL value in the join from either the Column or Operand tables. This is known as an outer join. After you choose the join, click on OK or Apply to initiate the query.

A bad join can occur when you try to join two unrelated columns. For example, if you joined the axparts and axorders tables, you would

create a database query with no meaning. The join condition would be:

```
axparts.part_num = axorders.order_num
```

Although both columns may contain similar numbers, the numbers in the columns represent two different values. A join on these columns produces useless information.

## Other Joins

There are other types of joins besides the table joins on common columns. Other joins include:

- Non-equal join
- Outer join

### Non-equal join

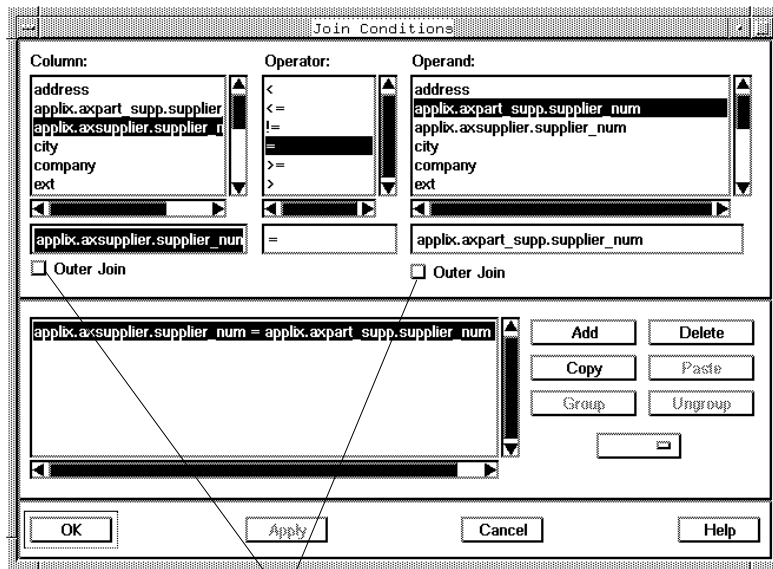
The joins described previously were based on the equality of the joined column. You can create a non-equal join of tables, comparing the differences of columns. You can choose a different comparison operator from the Operator list area in the Join Conditions dialog box. The option is equal by default, but you can choose less than, greater than, or not equal as the join condition.

### Outer join

When a standard equal join is created, rows in either of the join tables that do not match the join conditions are not included. An outer join includes all rows from a table, whether or not they match

corresponding rows in the joined tables. With an outer join, you can see the rows from the table that did not match the join condition.

The Join Conditions dialog box has Outer Join options to include the unmatched rows from either table in the query. Turn on the Outer Join option for either table to create an outer join.



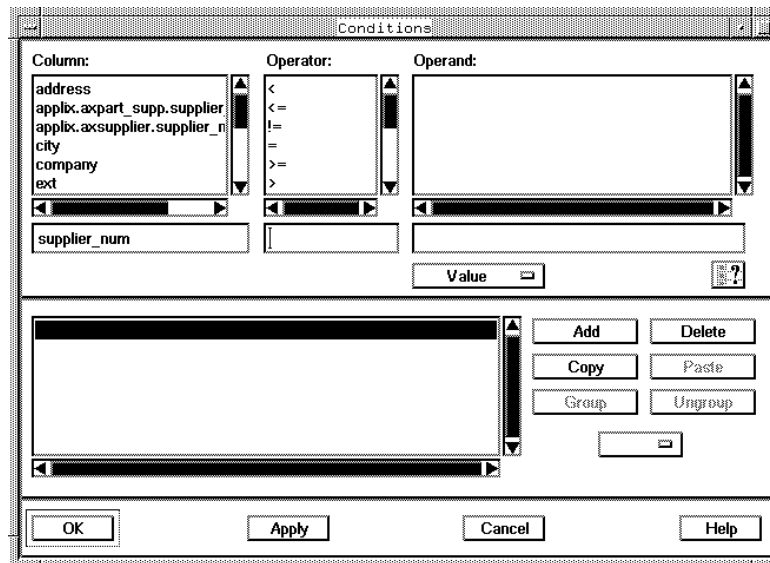
Outer Join option

### Ambiguous Column Reference

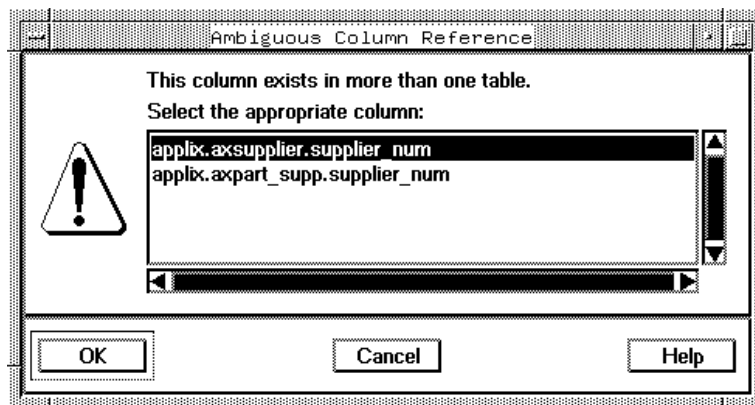
When you have a join between two or more tables, some columns in the tables may have the same name. When you set a query condition in the Conditions or Having dialog boxes, Applix Builder may be unable to distinguish which table column to use for a condition. For example, if you have a join on the axpart\_supp and axsupplier sample

tables, both tables contain a supplier\_num column. If you were setting a query condition with supplier\_num column, you would:

1. Choose **Query** → **Conditions**.
2. Type **supplier\_num** in the Column entry box.



3. Press **TAB**. The Ambiguous Column Reference dialog box is displayed.



Since the column exists in more than one table, Applix Builder cannot automatically choose a column. Choose the table column you want for the condition from the dialog box. The column you choose is placed in the Column entry box.

## Data Set Joins

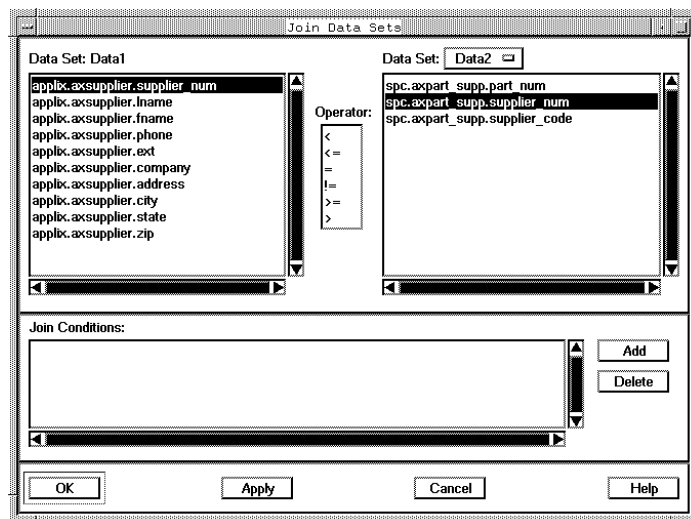
A join on data sets allows a logical join between separate databases, even databases from different vendors, located on different machines. A logical join of data sets allows you to change joins without edits in a Source data window.

Since the join is on different data sets, the performance is different from a join within a single data set. When information in one data set changes, such as current position or query conditions, the data set to which it is joined also needs to set current position or requery.

To create a data set join:

1. Choose **Application** → **Source** from the Builder main menu. The Data Source dialog box appears.

2. Choose a data source in the Data Source list area.
3. Click on **Join**. The Join Data Sets dialog box appears.



4. Choose a join column from the left Data Set list area.
5. Choose a data set with the Data Set option. The columns from the data set appear in the right Data Set list area.
6. Choose a join column from the right Data Set list area.
7. Choose a join condition operator from the Operator list area.
8. Click on **Add**.

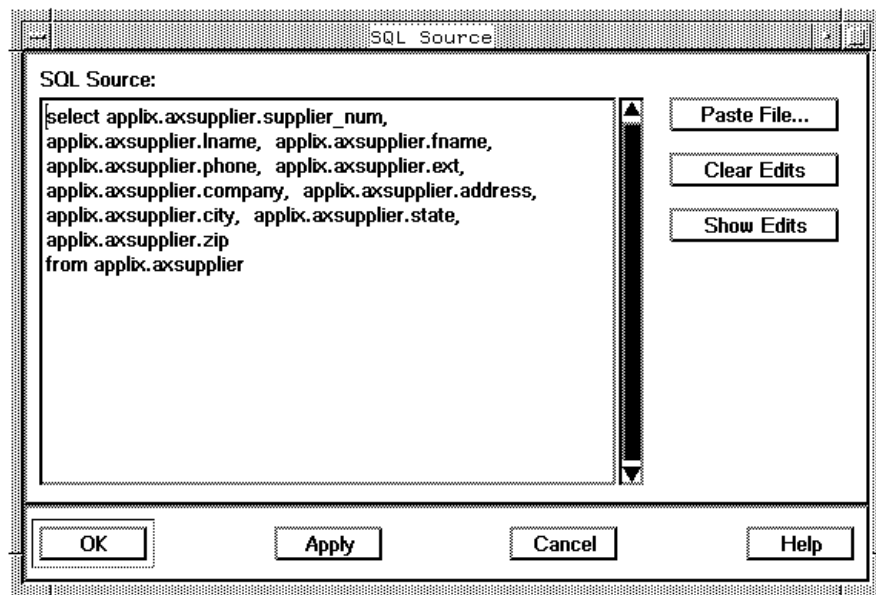
The join condition appears in the Join Conditions list area. Repeat steps 4-8 to create multiple joins on a data set.

## SQL Source

You can edit the SQL source code that Applix Builder generates from your database query with the SQL Source dialog box. After you choose a database server, you can use the SQL Source dialog box. You can add or delete parts of the SQL statement to refine your query. You can run SQL source code to create, modify, or drop tables in your database. You can even use the SQL source in a vendor's database application. Any changes you make to the SQL source are reflected in the Source data window. When you modify the SQL source code, however, the other Query menu options such as Choose Columns or Conditions are unavailable.

To edit the SQL source code:

1. Choose **Edit** → **SQL Source**.



2. Click in the source code area.

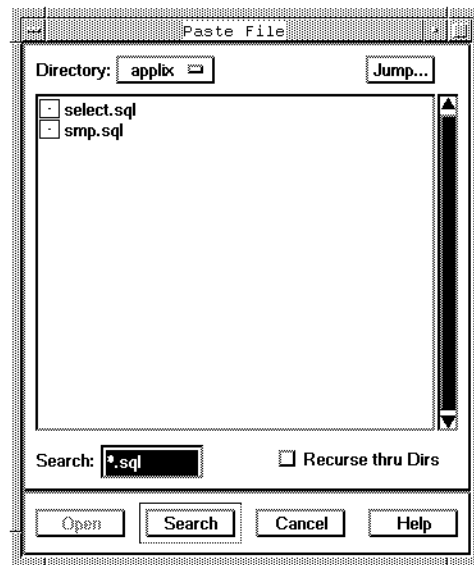
Use BACKSPACE to delete SQL source code. You can place the text cursor at any point in the SQL source and type additional code.

You can paste SQL source code in the current SQL source. Applix Builder only saves SQL select statements as part of an application file. SQL statements other than select, such as update and set, are not saved as part of the application file. SQL statements other than select are executed when you click on OK or Apply in the SQL Source dialog box, while an SQL select statement is executed only during a Source data window query.

**NOTE:** If you use SQL statements in the SQL Source dialog box that change database table information, such as update, you may change table information. You cannot undo or discard edits made directly to a database table with SQL statements.

To paste SQL source code in the current SQL source:

1. Click in the SQL Source list area in the SQL Source dialog box.
2. Click **Paste File** in the SQL Source dialog box.



3. Open a .sql file in the current directory, or use the Paste File dialog box to open a file in a different directory.

An SQL file is pasted at the cursor location in the SQL Source list area.

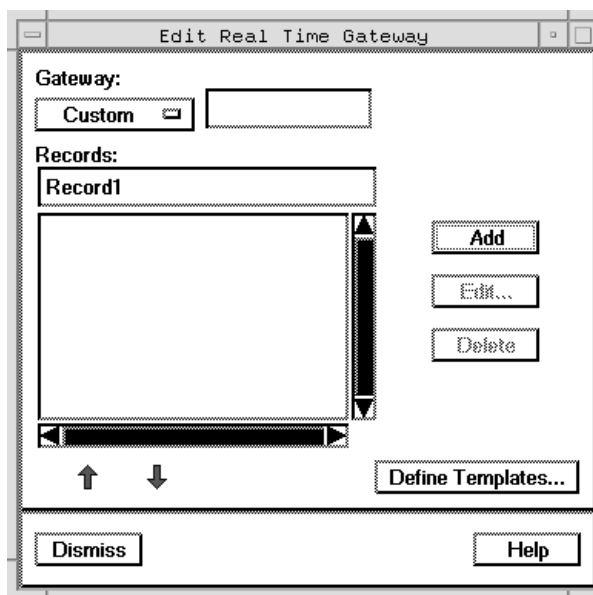
You can click on Clear Edits in the SQL Source dialog box to remove all edits and restore the original query SQL source code. You can only clear edits if you have not saved the file since the edits. You can click on Show Edits to restore any edits you made to the SQL source.

## Creating a Real Time Data Source

A Real Time data source provides a continuous stream of changing information, such as financial market information or data recorded by scientific instrumentation. You can use a Real Time data source to access, analyze, and publish large volumes of continually changing data.

To create a Real Time data source with the Data Source dialog box:

1. Choose **Application** → **Source** from the Builder main menu. The Data Source dialog box appears.
2. Type the data source name in the Data Source entry box.
3. Click on **Create**. The data source name appears in the Data Source list area.
4. Choose **Real Time** from the Source Type option.
5. Click on **Edit**. The Edit Real Time Gateway dialog box appears.



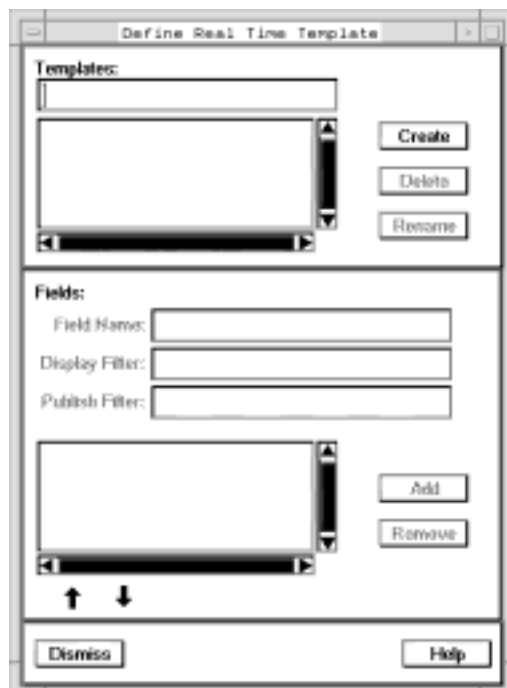
6. Choose the gateway with the Gateway option. For a custom gateway choose Custom and type the name of the gateway in the entry box.
7. Type the name of the record in the Records entry box. Use this name to reference the record in the Browser and Connector.
8. Click on **Add**. The record name appears in the Records list area.

After you create a record, you need to edit the record to define the record ID, the service used by the record, and the fields of interest in the record. You can define this information individually, or for multiple records with a template.

## Defining a Template

A template contains one or more fields. You can use a template to define fields for multiple records, instead of defining individual fields and properties for each record. To define a template:

1. Click on **Define Templates** in the Edit Real Time Gateway dialog box. The Define Real Time Template dialog box appears.



2. Type the template name in the Templates entry box.
3. Click on **Create**. The template name appears in the Templates list area.
4. Type a field name in the Field Name entry box.

5. Type a display filter method or macro name in the Display Filter entry box.
6. Type a publish filter method or macro name in the Publish Filter entry box.

Filters manipulate the field values retrieved or contributed to a Real Time data source. See "Field Filters" later in this chapter for information on writing and using display and publish filters.

7. Click on **Add**. The field name appears in the Fields list area.
8. Repeat steps 4-6 to add more fields to the template.

When you are finished defining a template, click on **Dismiss** to dismiss the Define Real Time Template dialog box.

## Modifying a Template

You can make modifications to a template as requirements change. Changes to a template include:

- Removing a field
- Changing field order
- Deleting a template
- Renaming a template

## Removing a Field

When a field is not required or does not exist in a record you may want to remove the field. To remove a field:

1. Click on a template name in the Templates list area.
2. Click on the field name in the Fields list area.
3. Click on **Remove**.

When you are finished removing a field, click **Dismiss** to dismiss the Define Real Time Template dialog box.

## Changing Field Order



Shift Up,  
Shift Down  
icons

To change the field order, click on the field name in the Fields list area, then use the Shift Up and Shift Down icons to establish the field order.

When you are finished changing field order, click **Dismiss** to dismiss the Define Real Time Template dialog box.

## Removing a Template

When a template is not required you may want to remove it from the application. To remove a template:

1. Click on a template name in the Templates list area.
2. Click on **Delete**.

When you are finished deleting a template, click **Dismiss** to dismiss the Define Real Time Template dialog box.

## Renaming a Template

When the contents of a template change you may want to change the template name to reflect the changes. To rename a template:

1. Click on a template name in the Templates list area.
2. Type the new template name in the Templates entry box.
3. Click on **Rename**.

When you are finished renaming a template, click **Dismiss** to dismiss the Define Real Time Template dialog box.

## **Editing a Record**

A record is not complete until you edit the record to define the record ID, the service used by the record, and the fields of interest in the record. To edit a record:

1. Choose a record in the Records list area of the Edit Real Time Gateway dialog box
2. Click **Edit** in the Edit Real Time Gateway dialog box. The Edit Real Time Record dialog box appears.



3. Type the record ID in the Record ID entry box. The record ID is used to register the record with the gateway and is the actual name used by the gateway to retrieve record information.
4. Choose the service with the Service Name option. To use a custom service, choose Custom and type the service name in the entry box. If you are editing an RTSQL record then service information is not required in this dialog box. See "Entering an RTSQL Query" later in this chapter for information about RTSQL queries.

You can use a defined template of fields with a record, or you can assign individual fields. You cannot use a template and assign individual fields. See "Defining a Template" earlier in this section for information on defining a template of fields.

If you are using a template with a record:

1. Turn on the **Use Template** option.
2. Choose the template from the Use Template option menu. The fields appear in the Fields list area.
3. Click **Dismiss** in the Edit Real Time Record dialog box.

If you are assigning individual fields to a record:

1. Type a field name in the Field Name entry box.
2. Type a display filter method or macro name in the Display Filter entry box.
3. Type a publish filter method or macro name in the Publish Filter entry box.

Filters manipulate the field values retrieved or contributed to a Real Time data feed. See "Field Filters" later in this chapter for information on writing and using display and publish filters.

4. Click **Add**. The field appears in the Fields list area.
5. Repeat steps 1-4 to add more fields to the record.
6. Click **Dismiss** in the Edit Real Time Record dialog box.

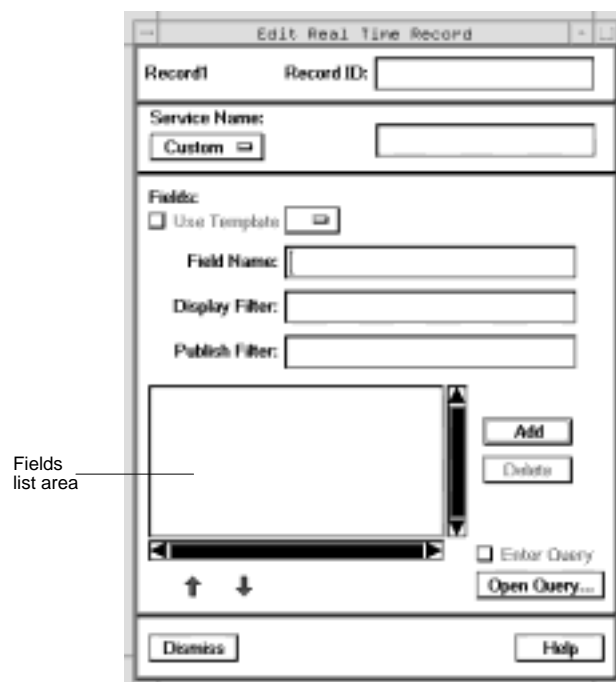
## Removing a Field

When a field is not required or does not exist in a record you may want to remove the field. To remove a field:

1. Click on the field name in the Fields list area.
2. Click **Delete**.
3. Click **Dismiss** in the Edit Real Time Record dialog box.

## Changing Field Order

To change the field order, click on the field name in the Fields list area, then use the Shift Up and Shift Down icons to establish the field order.



When you are finished changing field order, click Dismiss in the Edit Real Time Record dialog box.

## Field Filters

Filters manipulate the field values retrieved or contributed to a Real Time data feed. A filter is a macro or method available in the application.

A display filter takes a value from the Real Time data feed and returns a value for use in the application. You can use a display filter to format a value or substitute a string for a passed value.

The following display filter returns the character string NULL if the Real Time value is NULL. If the Real Time value is less than one the macro returns 0.

```
macro converter(value)
  IF IS_NULL@(value)
    RETURN("NULL")
  IF value < 1
    RETURN(0)
  ' If not NULL or less than 1 return value
  RETURN(value)
endmacro
```

A publish filter takes a value from the application and returns a value to publish to the Real Time data feed. You can use a publish filter to convert strings from an application to numerical values, or to otherwise prepare the data for contribution.

To use a filter, type the name of the method or macro in the Display Filter or Publish Filter entry box of the Edit Real Time Record or Define Real Time Template dialog boxes. A filter method should be a get method that returns a value or string. If you use a display filter macro, type the name of the macro in the Display Filter entry box, preceded by @. The @ before the macro name indicates the name is a macro, not a

method. For example, you would type the macro converter in the Display Filter entry box as:

@converter

The macro must be accessible by the application. If the macro is not in the object source, the application searches all other object sources, then macros in your ELF search path. A macro in the object source has precedence over all other macros with the same name.

## Entering an RTSQL Query

You can use an RTSQL record to retrieve database information in real time to use in your application. The record uses the rtsql gateway to implement the query. This section covers:

- RTSQL background information
- Trigger query
- Entering a query for a record

## RTSQL Background Information

An RTSQL query is an SQL query performed at timed intervals. The rtsql gateway, which runs on the client machine, launches a standard Applix database gateway on the database host. The database gateway enters a polling loop, with the following steps:

1. Performs a query.
2. Compares the results to the previous results.
3. Returns the results to the Spreadsheets document if there is a change.

4. If there is no change, the database gateway waits a pre-determined time before repeating the process.

You can set the time between queries (polling interval) in the `rtconfig` file, as described in "Setting the Polling Interval" in Chapter 7, "Applixware Components," of the *System Administrator's Guide*.

**NOTE:** An Applix Data database gateway must be installed on the database host machine. See "Starting `axnet` and Installing Database Gateways" in Chapter 6, "Applixware Communications," of the *System Administrator's Guide* for information on database gateway configurations.

An RTSQL query requires the following arguments:

host	The name of the computer on which the database server is running.
routing	The vendor gateway. Supply the vendor gateway name if you are using more than one Oracle database.
vendor	The name of the database vendor. The supported database vendors are Informix, Ingres, Sybase, and Oracle. The database names must be initial capitalized.
database	The database name.
server	The database server name, if required by the database.
trigger	A trigger query, described in detail later in this section.
return	The return query.

The return query is an SQL query which you use to retrieve information from a database. A simple query, such as

```
select * from applix.axline_items
```

selects all information from the `applix.axline_items` table. Use the SQL syntax supported by your database to create more advanced queries.

## Trigger Query

If your return query is complex, has a long execution time, or returns a great volume of data, you may not want to execute the query within the polling loop. You can create a trigger query to determine whether to execute a longer or more complex query. A trigger query is optional. If a trigger query is not supplied, the return query is used in the polling loop.

A trigger query is a smaller, faster SQL query, compared to the return query. A trigger query should return as few rows of information as possible to minimize the amount of information compared in the polling loop. A change in the trigger query executes the return query. The trigger query does not return information to the record. If the trigger query result set is larger than 500 rows, the return query is executed without comparing the trigger query results.

For example, you may have a complex query that retrieves the ten most recent deliveries from the `delivery` table. Instead of executing the query every time in the polling loop, you can check the `invoice_num` column to see if any new deliveries have arrived using the following trigger query:

```
select count(*) from applix.delivery.invoice_num
```

You can use a database timestamp as a trigger query, as long as the timestamp is not used in the return query.

Use the string `$ALWAYS` as the trigger query if you want the return query to execute every time in the polling loop.

You can take advantage of triggers within the vendor DBMS by using the trigger to set a flag for the RTSQL trigger query. If you have a DBMS trigger that increments a counter, for example, the RTSQL trigger query only needs to check the flag. This allows the trigger query to be very efficient.

You can use this technique even if you don't set triggers within the vendor DBMS. To do so, have the database update code increment a flag for the RTSQL trigger query.

If you are using an Applix Data document for service information, you must write a macro to assign the trigger query to the document. A typical trigger query macro looks like:

```
macro set_trigger
  var trigger_query
  trigger_query = "select count(*) from
    applix.delivery.invoice_num"
  dbase_set_trigger@(trigger_query)
endmacro
```

Use the ELF macro `dbase_set_trigger@` to assign a trigger query to an Applix Data document. The macro takes a string or an array of strings as its argument. Use the `dbase_set_trigger@` macro with no argument to remove a trigger query from an Applix Data document. Use the ELF macro `dbase_get_trigger@` to retrieve the trigger query assigned to an Applix Data document. The macro returns the query as an array of strings, even if the array contains only one item. To assign a trigger query to an Applix Data document after you write and compile a macro:

1. Open the Applix Data document in a Data window.
2. Choose **★ → Run Macro**. The Run Macro dialog box appears.
3. Type the macro name in the Macro Name entry box and run the macro.

4. Save and exit the Applix Data document.

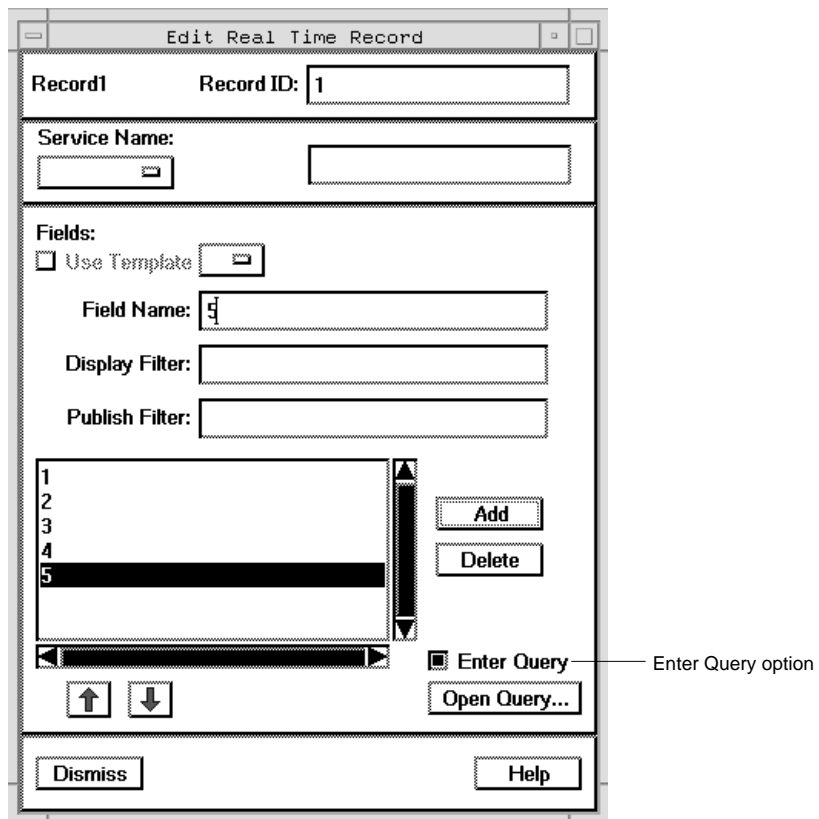
The trigger query is saved with the Applix Data document and is called during the polling loop.

## Entering a Query for a Record

To create an RTSQL record and enter a query for the record:

1. Choose rtsql as the gateway in the Edit Real Time Gateway dialog box.
2. Add and edit a record.
3. Type the row number for the returned data in the Record ID entry box in the Edit Real Time Record dialog box. The row numbering begins at 1
4. If you are using an Applix Data file as your query source, type the full path name, including .aq file extension, in the Service Name entry box.

If you want to enter a query, toggle on Enter Query in the Edit Real Time record dialog box.



5. Click **Open Query**. The RTSQL Query Info dialog box appears.

The image shows a dialog box titled "RTSQL Query Info". It contains a section labeled "Record1" with five input fields: "Host:", "Routing:", "Vendor:", "Database:", and "Server:". Below these are two larger text areas: "Trigger SQL:" and "SQL Query:". At the bottom of the dialog are two buttons: "Dismiss" and "Help".

6. Type the information, as required by your database, in the appropriate entry boxes, then click **Dismiss**.
7. Add the column numbers of returned data requested from the row in the Field Name entry box. Column numbering begins at 1.

You can use the RTSQL query as you would a standard Real Time record. You can access the information directly in the object, or connect the information to dialog box controls.

---

# 4 Using the Designer Tool

---

The Designer tool allows you to create dialog boxes that provide the interface to data sets or macros in the application. You can create, delete, edit, and design dialog boxes for your application. This chapter covers the following topics:

- Using the Designer dialog box
- Editing a dialog box
- Assigning a menu bar

## Using the Designer Dialog Box

Applix Builder applications depend on dialog boxes to convey information. Every application, by default, has at least one dialog box. Use the Designer dialog box to create and modify application dialog boxes. This section reviews the Designer dialog box features.

With the Designer dialog box you can:

- create a dialog box
- edit a dialog box
- rename a dialog box
- delete a dialog box
- assign a menu bar

### Creating a Dialog Box



Designer icon

To create a dialog box with the Designer dialog box:

1. Click on the Designer icon in the Browser window. The Designer dialog box appears.



2. Type the name of the dialog box in the Dialog Boxes entry area.
3. Click **Create**. The dialog box name appears in the Dialog Boxes list area.
4. Choose the type of dialog box with the Type option.

Four dialog box types are available for an application:

- A Normal dialog box is the standard dialog box type. Only one instance of a Normal dialog box is allowed at one time during application execution.
- A Main dialog box is the first dialog box that appears when you run an application. You can designate only one Main dialog box in an application.

- A Response dialog box suspends application action when it is displayed. Application action resumes when the dialog box is dismissed.
- A Multiple dialog box can have multiple instances at the same time.

When you create a dialog box a Dismiss push button is automatically inserted in the dialog box. The code in the Dismiss push button's object method source is a `clicked_event` programmed to close the dialog box. You can rename the push button, change the `clicked_event` in the object method source, or delete the push button.

You can associate a data set directly with a dialog box using the Designer tool. When you associate a data set with a dialog box, the Dialog Box Editor window automatically creates controls to display the data set information. See "Creating a Dialog Box with a Data Set" later in this chapter for more information.

## Renaming a Dialog Box

To rename a dialog box:

1. Click on the Designer icon in the Browser window. The Designer dialog box appears.
2. Click on a dialog box name in the Dialog Boxes list area.
3. Type the new dialog box name in the Dialog Boxes entry area.
4. Click on **Rename**.

The dialog box name is changed to the new name.

## Copying a Dialog Box

You can copy a dialog box from another application, or the contents of an Applixware dialog box from the Applixware Dialog Box Editor to an open Designer window.

To copy a dialog box between applications you can select the dialog box object and use the copy and paste editing options. When you copy and paste dialog boxes the copied dialog box retains all settings. The controls in the dialog box retain all their properties, as well as their object method sources. See "Adding and Deleting Objects" in Chapter 2, "Using the Browser Tool", for information on copying and pasting objects.

To copy the contents of an Applixware dialog box to an open Designer window:

1. Open the dialog box in the Applixware Dialog Box Editor.
2. Select all the controls in the dialog box.
3. Choose **Edit** → **Copy**.
4. Choose **Edit** → **Paste** in the open Designer window.

See Chapter 7, "Dialog Box Editor", in the *ELF User's Guide* for information on using the Applixware Dialog Box Editor. See "Cut, Copy, Paste, and Paste Dialog" later in this chapter for more information on dialog box control editing.

## Deleting a Dialog Box

All dialog boxes added with the Designer tool remain part of an application until removed. To delete a dialog box:

1. Click on the Designer icon in the Browser window. The Designer dialog box appears.
2. Click on a dialog box name in the Dialog Boxes list area.
3. Click **Delete**.

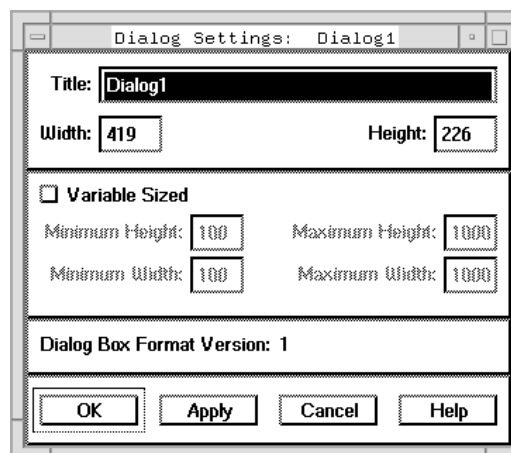
The dialog box is deleted. The dialog box name is removed from the Dialog Boxes list area.

## **Editing a Dialog Box**

You can edit a dialog box to add, delete, or change controls in the dialog box. To edit a dialog box:

1. Click on the Designer icon in the Browser window. The Designer dialog box appears.
2. Click on a dialog box name in the Dialog Boxes list area.
3. Click **Edit**. The dialog box appears in a Dialog Box Editor window.





The Title entry area shows the name for the dialog box. When you create a new dialog box, the name of the dialog box is placed in the entry area. To change the name of the dialog box:

1. Double-click in the Title entry area.
2. Type the new name in the entry area.

The initial width and height of the dialog box appear in the Width and Height entry areas. To change the width or height of the dialog box:

1. Double-click in the entry area.
2. Type the new dimensions in the entry area.

For a resizable dialog box, turn on the Variable Sized option to set the minimum and maximum resizing dimensions for the dialog box. Type the minimum and maximum widths and heights, in pixels, in the appropriate entry boxes. These settings apply to the actual use of the dialog box in an application. During dialog box editing the dialog box is resizable to any size.

## Dialog Box Controls

The dialog box controls provide the means for displaying and getting all types of information. While some information could be conveyed using any of several types of controls, there is usually one control that is best suited to presenting the information. By being consistent in your use of controls, you can speed the creation of dialog boxes and maximize the ease of use of your application.

The control types are:

Push button	A single button which performs an action programmed in the <code>clicked_event</code> method, such as the <code>Dismiss</code> button.
Toggle button	A toggle button can either be turned on or off. A toggle button control is used when you have a single item that can either be chosen or not chosen.
Entry area	An entry area is a text entry area where the user can type information. An entry area is useful for getting string information to be used with an application.
Option button	An option button offers a menu of options from which to choose.
Radio button	A radio button group is a group of items from which only a single item can be chosen at a time. Normally, radio button groups are a collection of logically associated items.
List Area	A list box is a group of items displayed inside a box. You can use vertical and horizontal scroll bars to scroll through entries if necessary. Some list areas allow more than one item to be chosen at

	a time, while other list areas allow only one item to be chosen at a time.
Edit box	An edit box displays a large amount of text that you can manipulate. You can add, edit, or remove information in an edit box. The text in an edit box automatically wraps, so the text will adjust to any edits. A vertical scrollbar allows you to move to different parts of the text in the edit box.
Scale	A scale is used to gradually increment and decrement values. A scale consists of a slider and sliding area. A scale can be used alone, or with an entry area to provide exact feedback on values.
Table	A table displays information in columns and rows. A table can have horizontal and vertical scroll bars so that you can move through the table information.
Canvas	A canvas provides a drawable area in the dialog box. Using a PenClass object and the canvas methods you draw text and graphical objects upon the canvas. Vertical and horizontal scrollbars allow you to move to different areas of the canvas.
Combo Box	A combo box offers a menu of options from which to choose. When you click on a combo box all choices remain visible until you choose an item or click elsewhere on the screen.
Tab	A tab is a specialized case of a layered panel. Use a tab to display different sets of controls in the dialog box without programming <code>is_hidden@</code> and <code>display@</code> attributes for each control.

Note that option buttons, radio buttons, and list areas are all used for displaying lists of options. The choice of which of these controls to use depends on your specific situation.

Radio button groups are appropriate for a small group of exclusive options, allowing only one selection from a group. List areas are ideal for very long lists of items or when you want to allow for the selection of more than one item. Option buttons are useful when you have a limited amount of space in your dialog box and when you have a small to medium-sized list of items.

If the list of options will change over time, you should use an option button or list area; adding to a radio button group could cause the radio button display to overwrite other controls or to exceed the dialog box borders.

**NOTE:** RowCol, Splitter, CrossTable, and Chart controls cannot be created directly in the Designer. Create a control in a dialog box and change the class with the Object Properties dialog box to create one of these type of controls.

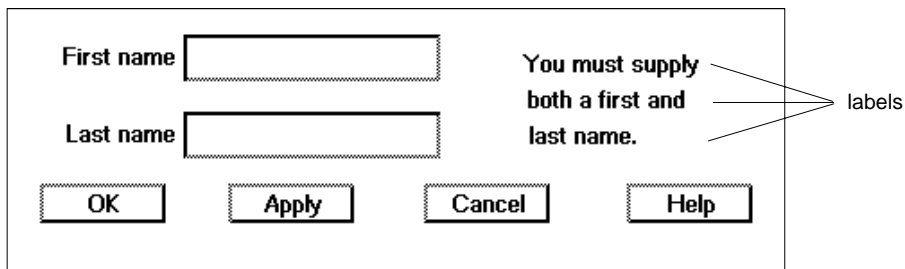
## Decorative Elements

There are three types of decorative elements that you can use in dialog boxes.

- Label
- Bitmap
- Panel/Line

## Label

A label is text that is not associated with a control. Labels are used primarily to provide instructions on the use of the dialog box or to provide information about the contents of a dialog box. The following shows examples of labels.

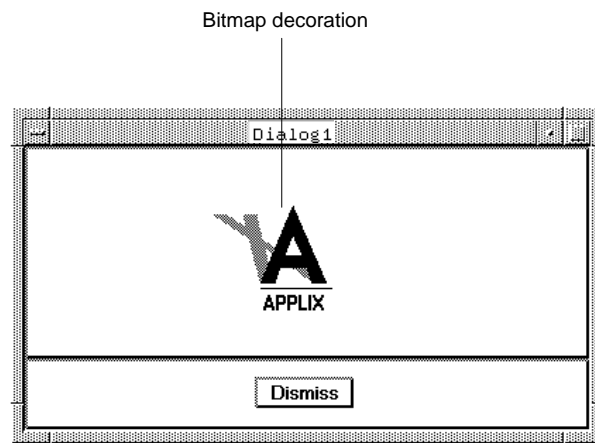


## Bitmap

A bitmap is a graphic element that you can use to decorate a dialog box. For example, on Applix Builder information message dialog boxes, a bitmap of the letter "i" is used to distinguish the dialog box as an information message.

Note that a bitmap decoration is different from a push button of type bitmap. A bitmap decoration is just a graphic element whereas a bitmap push button is a push button that is represented by a graphic element.

The following shows an example of a bitmap decoration.



## Panel/Line

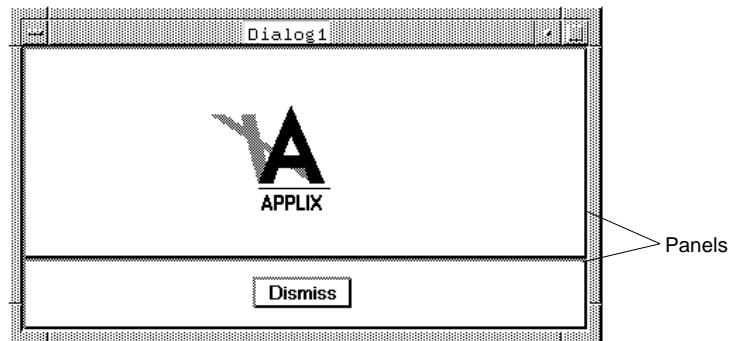
In addition to deciding what controls to use to represent information in a dialog box, you need to organize the controls to make the dialog box easy to use. The panel/line control is a design element that you can include in dialog boxes to visually group or separate dialog box elements.

A panel is a design rectangle that you can use to enclose several controls. You can specify the size and shape of the panel and the thickness of the lines that make up the panel. You can also indicate whether the panel should appear to be recessed or raised from the dialog box surface.

A line is a panel with a height of zero (0). A recessed panel has a negative thickness, while a raised panel has a positive thickness.

You can use lines as a design element in a dialog box to separate items in the dialog box.

The following shows an example of a dialog box containing two panels.



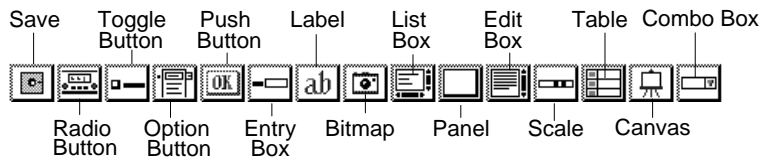
You can also add labeled or layered panels to a dialog box. A labeled panel is displayed with the panel title in the upper left corner of the panel.

Use a layered panel, also known as a tab control, to hide or display different sets of controls in the dialog box without programmatically setting the `is_hidden@` and `display@` attributes for each control.

## Adding a Control

To add a control, you can either:

- Choose a control from the Controls menu
- Click on an *ExpressLine* control icon

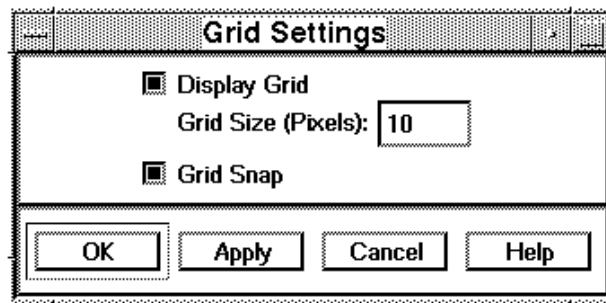


After you choose a control, click in the dialog box to place the control.

## Positioning Controls

You can position controls in the dialog box with your mouse. Click on the control and drag it to its new position. You can also use grids to position the controls in the dialog box.

Choose View → Grids. The Grid Settings dialog box appears.



You can turn on the grid, choose the grid size, and turn on the grid snap feature. The grid snap feature automatically positions the left corner of controls on the grids.

Choose View → Borders to turn on borders around dialog box controls. Borders are only visible in the dialog box editor; they are not part of

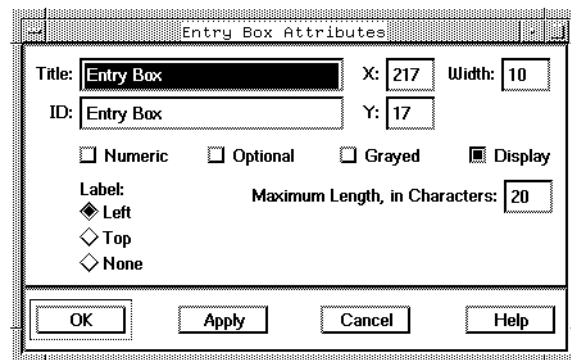
the dialog box. Use borders to show the mouse-sensitive area of dialog box controls.

## Control Attributes

Each control in your dialog box has an associated attributes dialog box. Double-click on a control and its associated attributes dialog box appears. Each control attributes dialog box contains entry areas for:

- Title
- Control ID
- X and Y coordinates in the dialog box

Different control attributes dialog boxes can contain additional entry areas and options for control features. For example, the Entry Box Attributes dialog box contains options for the entry box and display type, label position, and entry areas for the entry box width, and character length.



See the following sections for information on control attributes.

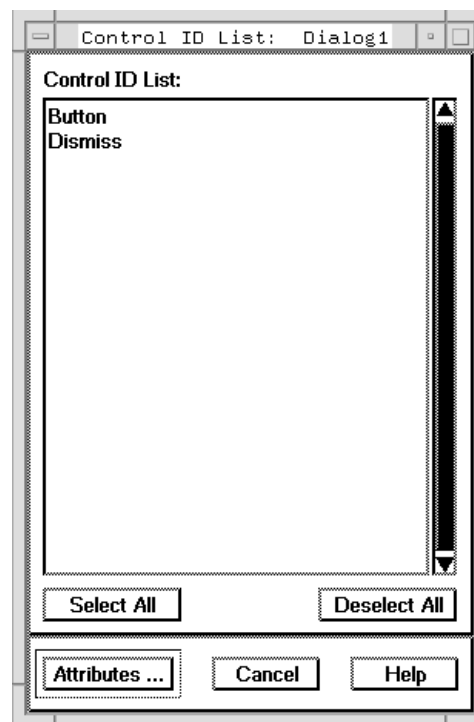
## Control IDs

When you create a control, a *control ID* is assigned to the control. The control's title is provided as a default control ID when you create the control. The ID is a string by which the control is referenced in the Object Browser. You can change the control ID in the control's attributes dialog box.

The control ID and the title do not need to match. Control IDs are useful when a control's title is long and you want to provide a shorthand way to refer to the control. For example, the title text for an entry area might be "Name of file to edit," so you might give the entry area control the ID "name." When you specify controls, be sure that each control ID in the dialog box is unique to assure correct dialog box functionality.

## Selecting Controls

You can select a control by clicking on the control in the dialog box. You can also choose Edit → Select and the Control ID List dialog box is displayed.

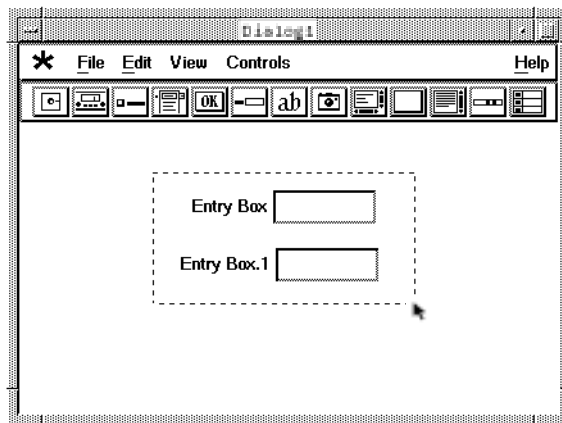


To select a control, you can click on a control ID in the Control ID List area, or you can click on **Select All** for all controls in the dialog box. Click on **Attributes** to change the attributes of selected controls. The Control ID List dialog box is useful for selecting hidden controls in the dialog box.

### Selecting Multiple Controls

You can select multiple controls in a dialog box using one of the following methods.

- Press and hold down the mouse button and drag the mouse pointer. A selection rectangle appears as you drag the mouse pointer as shown in the following:



All objects enclosed within the selection rectangle are selected when you release the mouse button.

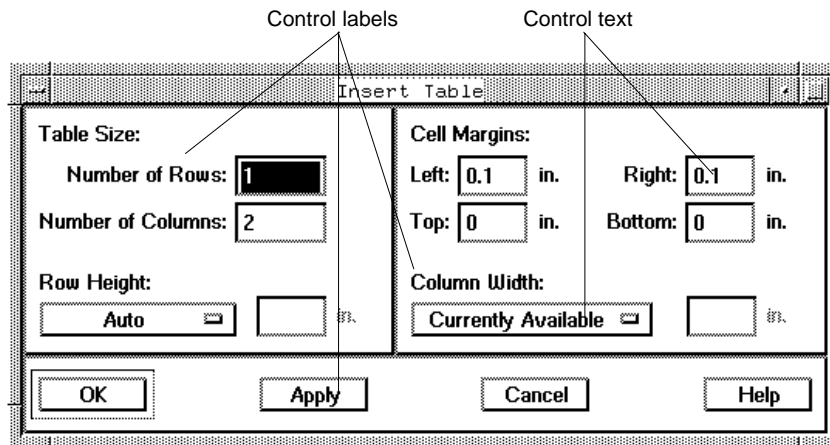
- Select each item individually. Select the first object by clicking on the border of the object. Select additional objects by holding down the CTRL key while clicking on the additional objects. Holding down the CTRL key enables you to select a new object without deselecting other objects.

You can combine the two selection methods. For example, you can use a selection rectangle to select objects located near each other, then use CTRL-click to select another object that is apart from the other objects.

## Character Settings

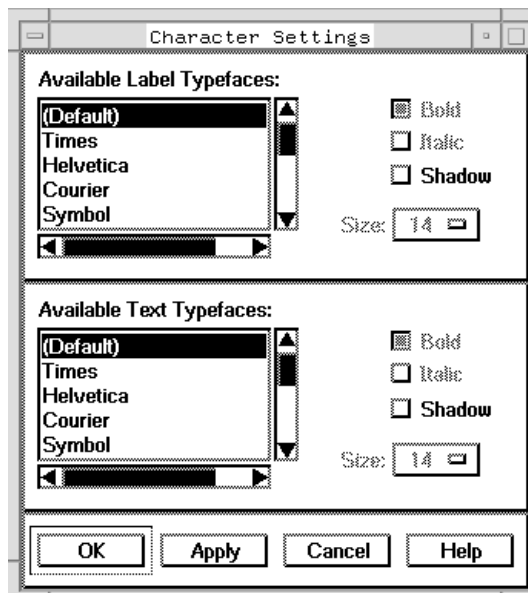
Controls appear in a dialog box with the default label and text settings. You can change the character settings for control labels or text in a dialog box with the Character Settings dialog box.

A label is the text that appears as the title of a control, for example, an entry box label or a push button name. Text is the text that appears in the control, for example, the text typed in an entry area or the choices in an option button.



To change a control's character settings:

1. Choose the control in the dialog box.
2. Choose **Edit** → **Character Settings**.



3. Change the label and text character settings in the Character Settings dialog box.

The options in the Character Settings dialog box are grayed if the option is unavailable. For example, a push button has a label, but it does not have text. When you choose a push button in a dialog box, the Text options in the Character Settings dialog box are grayed.

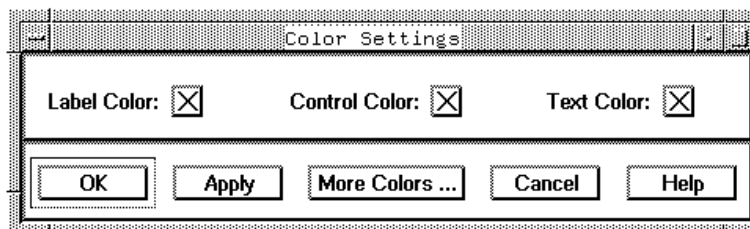
## Color Settings

Controls appear in a dialog box with the default color settings. You can change the color settings for controls, control labels, or text in a dialog box with the Color Settings dialog box. Colors can be set individually for each control, allowing you to highlight controls in a dialog box.

**NOTE:** You need a color monitor to edit colors with the Color Settings and Edit Color Palette dialog boxes.

To change a control's color settings:

1. Choose the control in the dialog box.
2. Choose **Edit** → **Color Settings**.

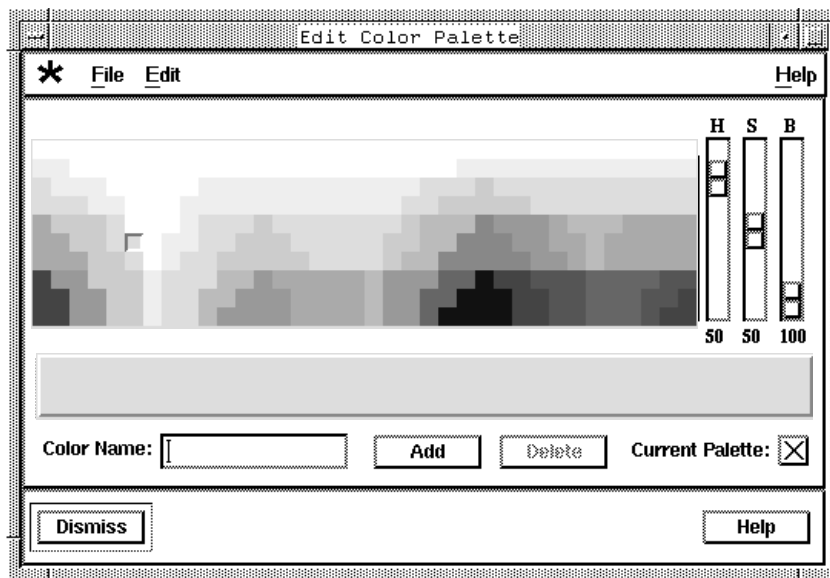


3. Change the color settings in the Color Settings dialog box.

Control, label, and text color are initially set to the default color. Choose \* → Color Settings and click on the Color option to change your color preference setting. When you click on a color option, the available colors in the color palette appear. You can choose a color from the color palette, or create your own color with the Edit Color Palette dialog box.

The Edit Color Palette dialog box allows you to add or delete colors from the current color palette. To change the current color palette from the Color Settings dialog box:

1. Click **More Colors**.



2. Change the color palette.

The Color area shows the available colors you can add to the palette. You can click in the Color area to choose a color, or you can use the HSB scale options to adjust the hue, saturation, and brightness of the color. The HSB settings for the current color in the Color area appear beneath the scales, and the current color appears in the Current Color area. If you want to add a color to the current color palette:

1. Click on a color in the Color area or use the HSB scale options to set a color.
2. Type a name in the Color Name entry area.
3. Click on **Add**.

The color is added to the current color palette and is immediately available in the Color Settings dialog box for control, label, and text colors.

If you want to remove a color from the current color palette:

1. Choose the color with the Current Palette option.
2. Click on **Delete**.

You can save a color palette as a file, allowing you to use the same color palette in different dialog boxes. After you have saved color palettes, you can choose File → Open from the Edit Color Palette dialog box to load a color palette in the current dialog box.

## Deleting Controls

To delete a control from the dialog box:

1. Click on the control in the dialog box.
2. Choose **Edit → Delete Selected**.

You can delete more than one dialog box control at one time with the Control ID List dialog box. To delete multiple controls at one time:

1. Choose **Edit → Select**.
2. Choose the controls in the Control ID List dialog box. The controls appear selected in the dialog box.
3. Choose **Edit → Delete Selected**.

Note that once you delete a control, it is not recoverable.

## Cut, Copy, Paste, and Paste Dialog

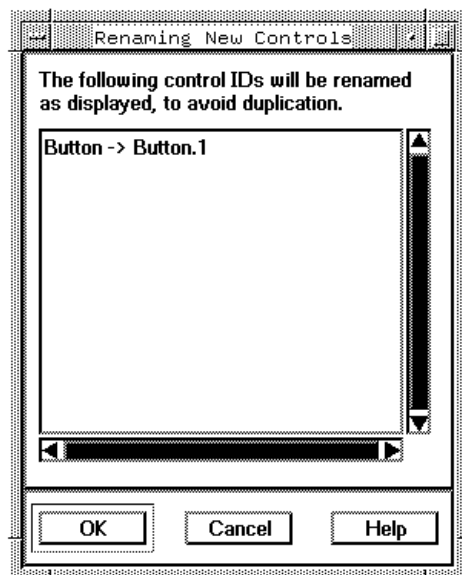
The Dialog Box Editor window contains cut, copy, and paste features to handle control editing.

To remove selected controls from the dialog box and move them to the clipboard, choose Edit → Cut. To put a copy of selected controls onto the clipboard, choose Edit → Copy. Material remains on the clipboard until you overwrite it by cutting or copying other material.

Use Edit → Paste to put material from the clipboard into the dialog box. You can repeatedly paste the same material, since the clipboard remains intact until more material is cut or copied onto it.

To activate this option, you must select at least one control.

The Dialog Box Editor window contains safeguards to avoid the duplication of control names. When you copy a dialog box control with Edit → Copy, then choose Edit → Paste, the Renaming New Controls dialog box appears.



The Renaming New Controls dialog box also appears when you choose Edit → Paste Dialog, and at least one of the controls in the pasted dialog box has the same name as a control in the current dialog box.

The original control ID appears in the list area with the proposed renamed control ID. Click on OK to accept the proposed renaming of the controls, or click on Cancel to abandon the paste.

To paste the contents of an existing dialog box file into the current dialog box choose Edit → Paste Dialog. The Paste Dialog Box dialog box appears.

The dialog box shows all the dialog box files and directories in the current directory. Double-click on the document you want to open, or change directories to a different dialog box.

## **Undo**

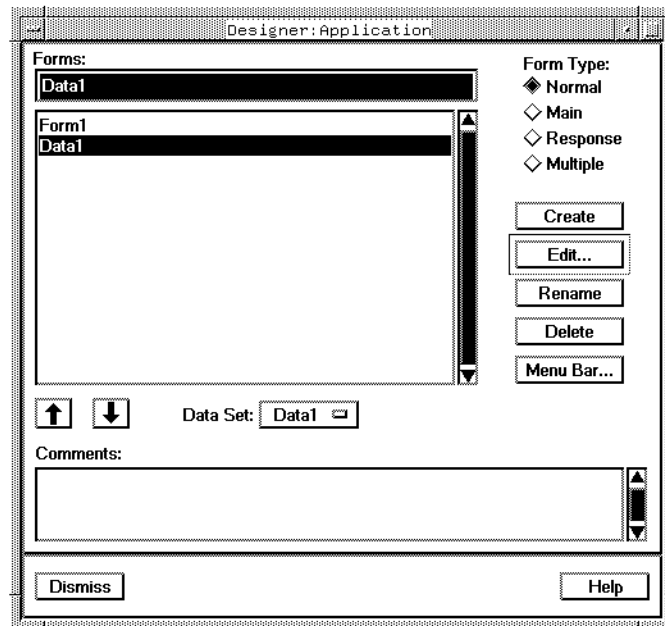
The Dialog Box Editor window has a one-level undo option to cancel the most recent edit action. To cancel an edit action, choose Edit → Undo.

## **Creating a Dialog Box with a Data Set**

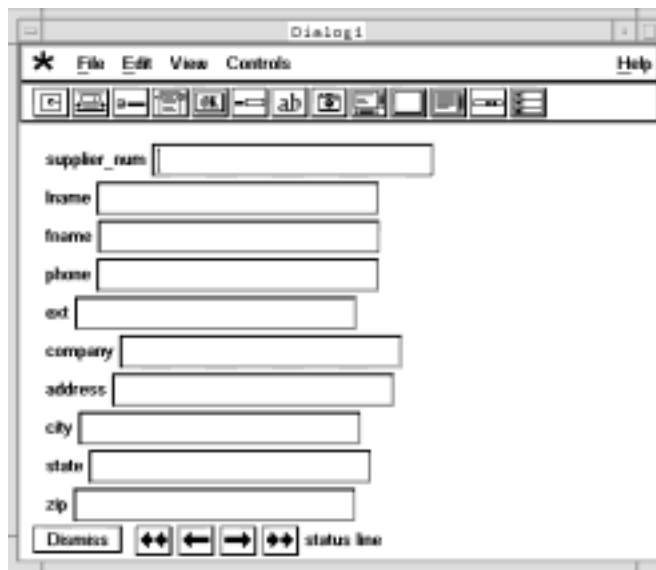
You can associate a data set directly with a dialog box using the Designer tool. When you associate a data set with a dialog box, the Dialog Box Editor window automatically creates controls to display the data set information. The dialog box cannot have children if you want to automatically create controls with the data set. Add menu bars and additional controls after you associate the data set, create the dialog box, and edit the dialog box.

To create a dialog box with an associated data set:

1. Choose **★** → **Designer** from the Builder main menu.
2. Type the name of the dialog box in the Dialog Boxes entry area.
3. Click on **Create**. The dialog box name appears in the Dialog Boxes list area.
4. Choose the type of dialog box with the Type option.
5. Choose a data set with the Data Set option. The option is only available if at least one data set exists for your application.



6. Click on **Edit**.



The dialog box appears in a Dialog Box Editor window. The Dialog Box Editor window automatically creates controls to display the data set information, with an entry area for each column of the data set and push buttons for accessing the database information.

The created dialog box is fully functional and ready to run. You do not need to connect the dialog box controls to the data set with the Connector tool. You can move, modify, or delete the controls to change their default layout.

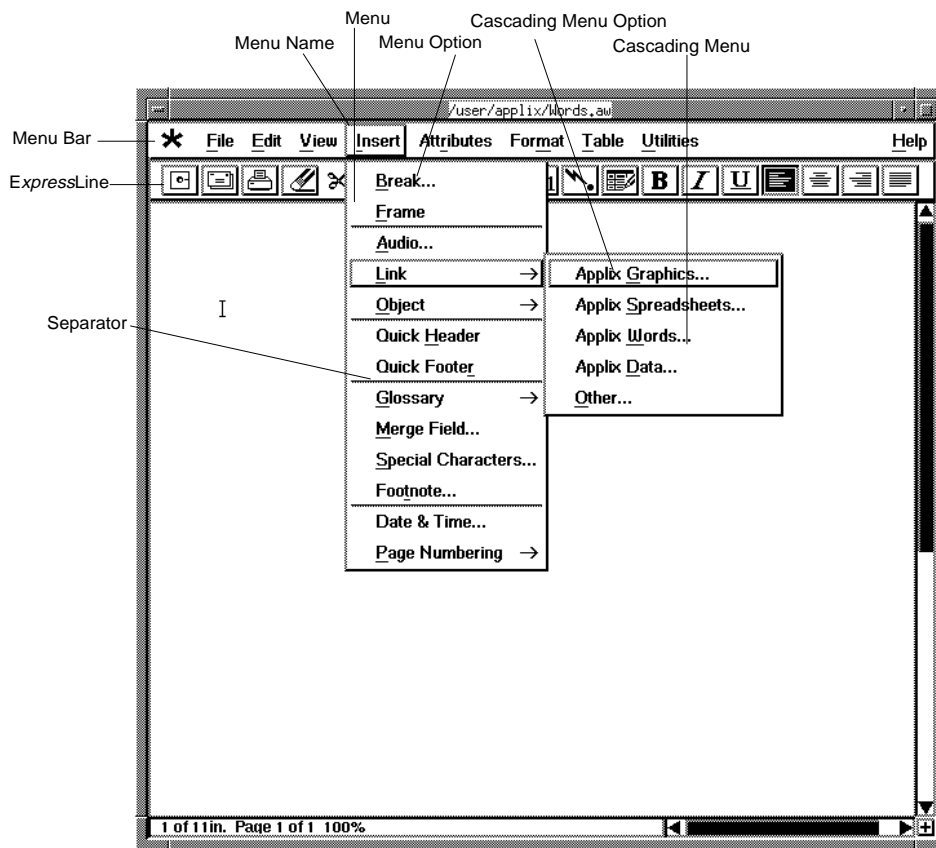
## Assigning a Menu Bar

After you create and edit a dialog box you can assign a menu bar. The Menu Bar Editor is accessible from the Designer dialog box. You can design and arrange the Menu Bar options, separators, and *ExpressLine* using the Menu Bar Editor. As you design more sophisticated applications and dialog boxes, you can mold the presentation of the application into a format that best suits your individual working style and needs.

### Identifying Parts of the Menus

The following shows the menu parts you can manipulate using the Menu Bar Editor:

## Assigning a Menu Bar



Dialog box menus contain the following elements:

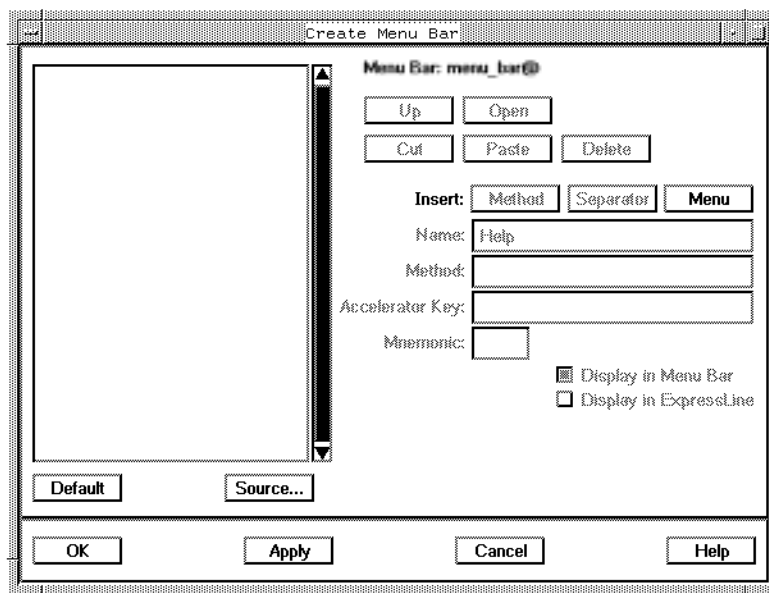
- |              |                                                                                                                                                |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| Menus        | The top-level item that appears in the horizontal Menu Bar. You can press the mouse on menu names to display a pull-down list of menu options. |
| Menu options | Menu options are items in pull-down menus that perform actions when they are chosen. For                                                       |

	<p>example, Save is a menu option on the File menu that performs the action of saving a document.</p>
Cascading menus	<p>A cascading menu is a sub-menu that appears in a pull-down menu. When you drag the mouse to the right of a cascading menu name, a list of menu options appears. Cascading menus have a right-pointing arrow.</p>
Separators	<p>The horizontal line used on a menu to visually group menu options together is called a separator. On the <i>ExpressLine</i>, separators appear as blank spaces.</p>
Mnemonics	<p>The underscore character shown in the Menu Bar and in the menus themselves are another keyboard shortcut available called <i>mnemonics</i>. Mnemonics are used only in the Motif window manager. Use the ALT, Meta, or COMPOSE CHARACTERS key, then press the Menu Bar mnemonic, then the menu option mnemonic to run an option. The mnemonic for the ★ menu is “a.” Be sure the mouse pointer is <i>not</i> within a menu when you use the mnemonic.</p>

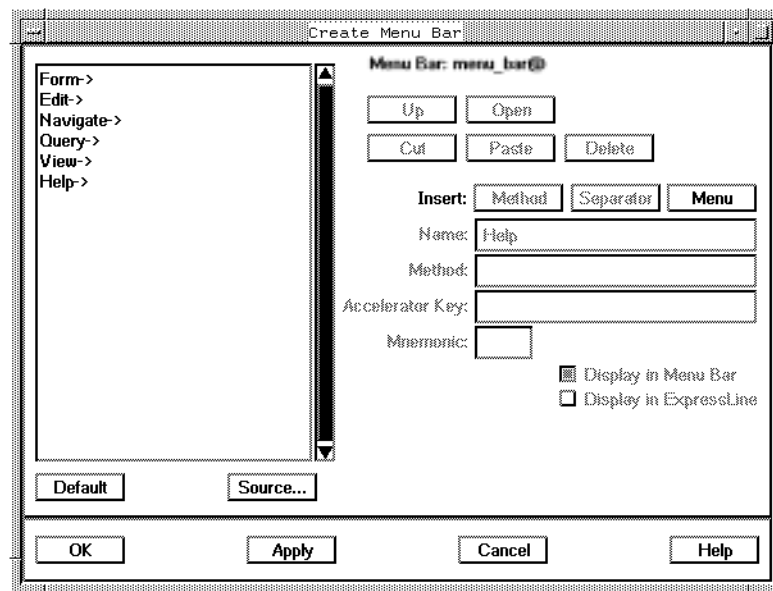
## Starting the Menu Bar Editor

To start the Menu Bar Editor:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.



The Create Menu Bar dialog box appears. If the dialog box has a menu bar assigned to it, the menu items appear in the list area, otherwise the list area is empty.



The menu item list area on the left shows each menu item as it appears on the application's Menu Bar.

Note that some Menu Bar Editor controls appear grayed. Certain controls only apply when you have selected a menu item from the display or you are displaying the contents of a cascading menu. If a control does not apply to the current state of the Menu Bar Editor, it appears grayed and cannot be chosen.

## Displaying Menu Items

When you first display the Menu Bar Editor, the list area shows the menus that are on the Menu Bar for the current application.

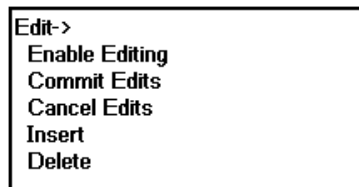
An arrow (->) in the list area after a menu name indicates that the menu item calls a pull-down or cascading menu. An ellipse (...) after

the name in a menu or in the list area indicates that the menu item calls a dialog box.



To update the list area so it displays the pull-down menu options for a particular menu, double-click on the menu you want to display. Or, you can:

1. Single-click on the menu you want to display to select it. The menu name is highlighted.
2. Click on **Open**.



Default Edit menu expanded

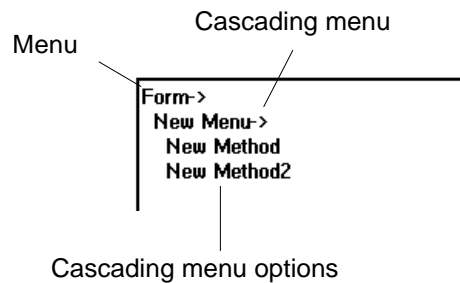
## Displaying Cascading Menu Options

Cascading menus are identified by an arrow after the menu name in the menu item list area. To update the list area so it displays the menu options in a cascading menu, double-click on the cascading menu option or:

1. Click on the cascading menu option to select it.  
The menu option name is highlighted.
2. Click on **Open**.

The cascading menu options for the menu are displayed in the menu item list area.

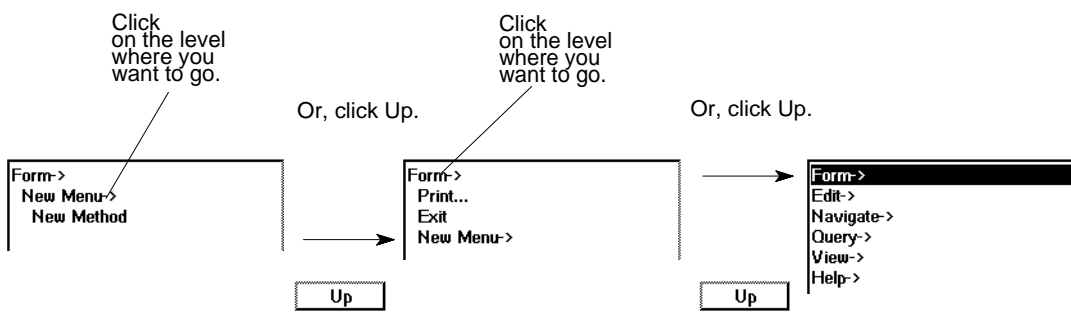
When you display the pull-down options for a menu, the name of the menu is shown at the top of the list and each of the pull-down menu options are indented. When you display the options in a cascading menu, each level is further indented as shown in the following:



**Figure 4-1 Cascading Menu Options in Menu Bar Editor**

## Moving Up through Menu Displays

Click on the Up button to move the list area display back up one level through the menu structure from the level you are currently at. Or, you can simply click on an item that is at the level where you want to go.

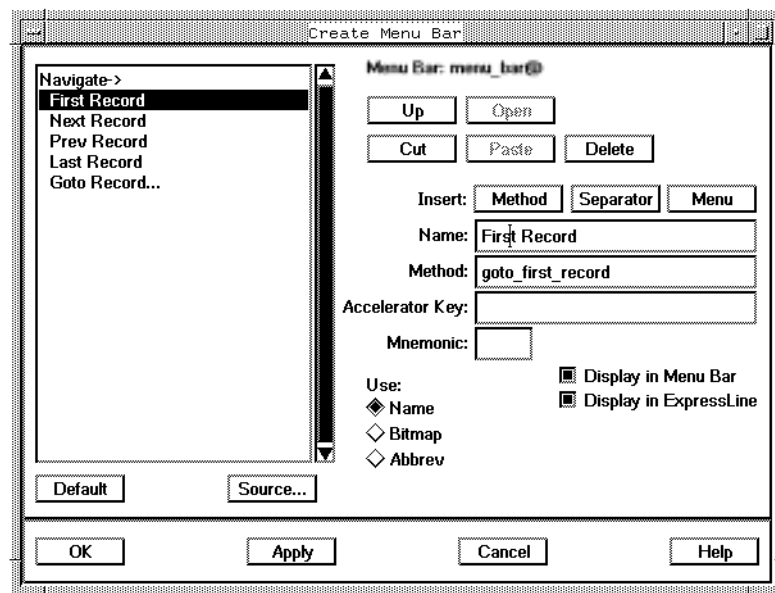


**Figure 4-2 Moving Up through Menu Displays**

## Displaying Information for a Menu Item

Once you are at the appropriate level for the menu item you want to customize, select the menu item you want to work on by clicking on it.

When you select a menu item (menu, cascading menu, menu option, or separator) from the menu item list area, the Menu Bar Editor displays information pertaining to that menu item as shown in the following example.



The information displayed includes:

**Name**                      The name of the menu item selected. This is the name that appears in the menu. Menu items that bring up dialog boxes are identified by ellipses (...) at the end of the menu name.

A name preceded by @, in the form @name, is a get method. Use a get method to display a variable string in the menu, such as the state of a toggle button. The get method must be in the menu bar object's source, and the method must return a string.

**Method**                      The name of the method that is called by this option appears here. When you add a new menu option, edit the method\_to\_invoke default name to the name of the method to use.

	<p>Method is used only by menu <i>options</i>, so nothing appears here when the item selected is a menu, cascading menu, or separator.</p>
Accelerator Key	<p>Displays a keyboard shortcut used to call the menu option. Only menu options can have accelerator keys, so nothing appears here when the item selected is a menu, cascading menu, or separator. Note that not all menu options have an accelerator key.</p>
Mnemonic	<p>Display the single character used to call this option when the ALT, Meta, or COMPOSE CHARACTER key is pressed. Mnemonics must exist as a character in the menu or option name. An underscore shows in the application when a mnemonic is defined for this option. All menu options have a mnemonic.</p>
Display in Menu Bar	<p>When this option is turned on, the menu item appears in the Menu Bar. When this option is turned off, this menu item is not visible with its menus. You may make the item available in the <i>ExpressLine</i> (see the next option), or make it available only through the accelerator key. Menu items can appear in both the Menu Bar and the <i>ExpressLine</i> at the same time.</p>
Display in ExpressLine	<p>When this option is turned on, the menu item appears in the <i>ExpressLine</i>. Items available in the <i>ExpressLine</i> can be shown as the Name (the menu name), Abbreviation (a shortened form of the menu name), or as a Bitmap. For a Bitmap, you must supply the file name of the bitmapped or digitized image. Menu items can appear in both the Menu Bar and the <i>ExpressLine</i> at the same time.</p>

Name	When this option is highlighted, the menu item appears in the <i>ExpressLine</i> with its menu name as the label on the button. This option is available only when Display in <i>ExpressLine</i> is selected.
Bitmap	When this option is selected, the menu item appears in the <i>ExpressLine</i> as a bitmapped image. You must enter the file name (without its extension) in the Bitmap entry area. The icon appears to the right of the entry area when an appropriate file name is entered here. This option is available only when Display in <i>ExpressLine</i> is selected. The bitmap is located in your ELF search path or is loaded as an application resource.
Abbreviation	When this option is selected, the menu item appears in the <i>ExpressLine</i> as the text entered in the Abbreviation entry area. Enter the text you want to appear there once this option is selected. This option is available only when Display in <i>ExpressLine</i> is selected.

## Hidden Menus

Most menus that you create are designed to appear in either the Menu Bar or the *ExpressLine*. You make a newly created menu item appear by turning on either the Display in Menu Bar option or the Display in *ExpressLine* option, or both. However, sometimes you may want to create an entire menu that is accessible only through the keys. These are called *hidden menus*. By default, the Keys menu found in every Applixware application is a hidden menu. The Keys menu gives you access to many functions by means of the keyboard only.

If no menu options have been defined for a menu, or if *none* of the options in the menu have Display in Menu Bar or Display in *ExpressLine* turned on, then the menu is *hidden* and is not displayed on the Menu Bar for an application. You can place groups of related

keyboard-based menu options in hidden menus so that you can easily locate them using the Menu Bar Editor if you need to change or delete them.

## The Keys Menu

Each Applixware application includes a Keys hidden menu that contains keyboard shortcuts for performing operations such as editing text, moving the text cursor, and scrolling a window.

The information for any option in the Keys menu can be changed the same way as information is changed for any other menu option. For example, you can change the accelerator key assigned to an option in the Keys menu.

If you turn Display in Menu Bar or Display in ExpressLine on for *any* of the Keys menu options:

- The Keys menu is no longer hidden and appears on the application's Menu Bar.
- The option that had Display in Menu Bar or Display in ExpressLine turned on appears on the application's newly displayed Keys menu.

If you add menu options that you want to be keyboard-based only, you might want to add them to the Keys menu so that all such menu options are grouped in one place and are easy to find should you want to change them.

## Changing Menu Items

You can use the Menu Bar Editor to change the name, accelerator key, Menu Bar location, ExpressLine location, bitmap, abbreviation, or macro call for existing menu items.

To change any of these attributes, first display the desired menu item in the Menu Bar Editor by moving to the item as described in “Displaying Menu Items” earlier in this chapter.

### Changing a Menu or Option Name

You can change the name of a menu or change the name used to identify a menu option. The menu and option names appear as text on the menus and need not be similar to the name of the macro that activates the option. To change the name for a menu option, select the appropriate menu option and type the new name in the Name field. To change a menu name, select only the menu itself and change its name in the Name entry area. Spaces are permitted in menu and option names.

A name preceded by @, in the form @name, is a get method. Use a get method to display a variable string in the menu, such as the state of a toggle button. The get method must be in the menu bar object’s source, and the method must return a string.

### Changing the Method Call

To change the method called by a menu option, select the appropriate option and type the name of the new macro in the Method entry area. The macro need not exist to be entered here; however, if the application cannot find it when the menu option is chosen, the menu option will not work.

### Changing Accelerator Key Assignments

You can change accelerator key assignments to suit your needs. Some Applixware menu options do not include accelerator keys. If you find that you frequently use a menu option that has not been assigned an accelerator key, you can add an accelerator key for the option.

Each application stores a separate set of accelerator keys so you can use the same accelerator key in different applications to perform different actions.

To change the accelerator key assignment for a macro option, select the appropriate option and type the new accelerator key in the Accelerator key entry area. Use ! for SHIFT, ^ for CONTROL, and \* for ALT or Meta.

If you specify an existing accelerator key, Applixware accepts any other changes you've made on this dialog box, but does not overwrite the existing accelerator key assignment. A message displays indicating that the key is already used. Assign this menu option a new key.

## Changing the Menu Option Accessibility

Methods can be accessed in four different ways with a menu bar. Depending on the way you want to use this method, you can change its access so it is available:

Only in Menu Bar	Click on Display in Menu Bar with nothing else selected.
Only in ExpressLine	Click on Display in ExpressLine with nothing else selected.
In both Menu Bar and ExpressLine	Click on both Display in Menu Bar and Display in ExpressLine.
Only by keys	Enter an accelerator key with nothing else selected.

With the appropriate menu option selected, you can change these aspects of the option as needed. Remember—to hide a menu, *all* options must have both Display in Menu Bar and Display in ExpressLine turned off.

## Changing the Order of Menu Items

You can change the order in which menu options or menus themselves are presented. See "Moving Menu Items" later in this chapter for details.

## Adding Menu Items

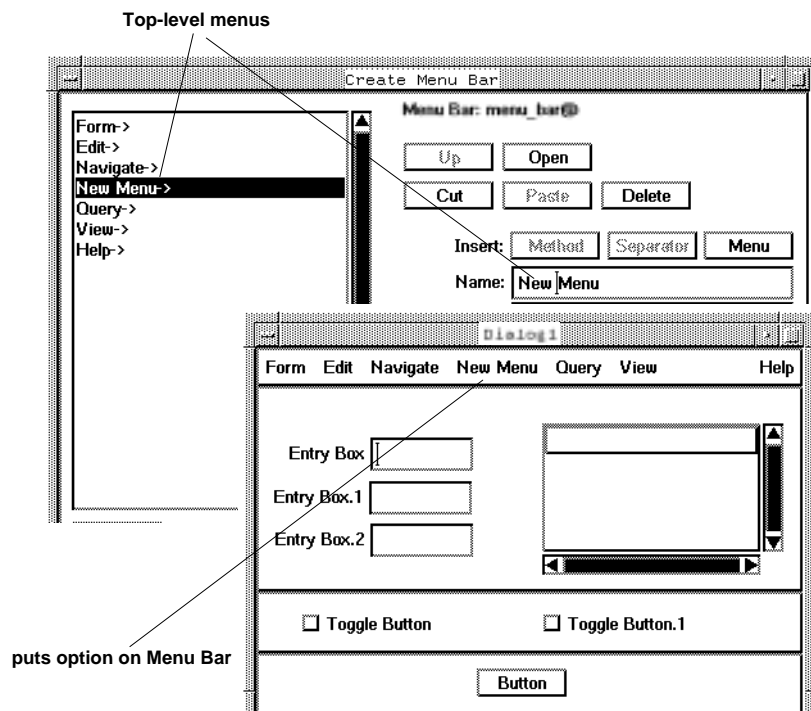
The Menu Bar Editor Insert options allow you to add menus to a Menu Bar, add menu options (that call methods) to pull-down or cascading menus, or add separators to divide menu options.

You can determine the order in which menu options and menus are presented by adding them in the appropriate place in the Menu Bar Editor list area:

- |                                    |                                                                                                                                                                    |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Add item within the list           | Select the menu item you want to come <i>after</i> the new menu item. For example, to insert a new menu between the Form and Edit menus, you select the Edit menu. |
| Add item to the top of the list    | Select the first item in the list. For example, to insert the new item before the Exit option on the Form menu, you select the Exit option first.                  |
| Add item to the bottom of the list | Add the new option. This inserts the new item at the bottom of the list and presents it last in the Menu Bar, ExpressLine, or menu.                                |

## Adding Menus

To add a new menu to the Menu Bar of any dialog box, be sure you display the top-level of menus in the Create Menu Bar dialog box. The following shows the result of adding a top-level menu:



To add a top-level menu to a dialog box's Menu Bar:

1. Choose **★ → Customize Menu Bar** in the appropriate application.
2. Select the item in the list area that will be *after* the menu option you want to add.

In the example shown in the previous screen, you would select the Query menu. If you want the new menu to be the last menu in the list, do not select anything in the list area.

3. Click on **Menu**.

When you click on Menu, the name `New_Menu->` appears in the list area. The Name entry area shows the name as `New_Menu`.

4. Type a name for the menu in the Name entry area.

To overtype New\_Menu, double-click in the entry area, then type the new name. The text you type appears in the Menu Bar to identify this menu. Names can include spaces.

5. Click **OK** or **Apply** to store the change and update the list area.

At any time, you can update the list area by selecting a different menu item from the display, or moving from then returning to the current menu.

Before a newly-added menu can become visible on the application's Menu Bar, you must add at least one menu option that is not a hidden menu. See the next section, "Adding Menu Options" for details.

Since macros and accelerator keys do not apply to menus, those entry areas are grayed.

Note that an arrow (->) is automatically appended to the menu name in the menu item list area to indicate that the item added can display a pull-down menu or a cascading menu.

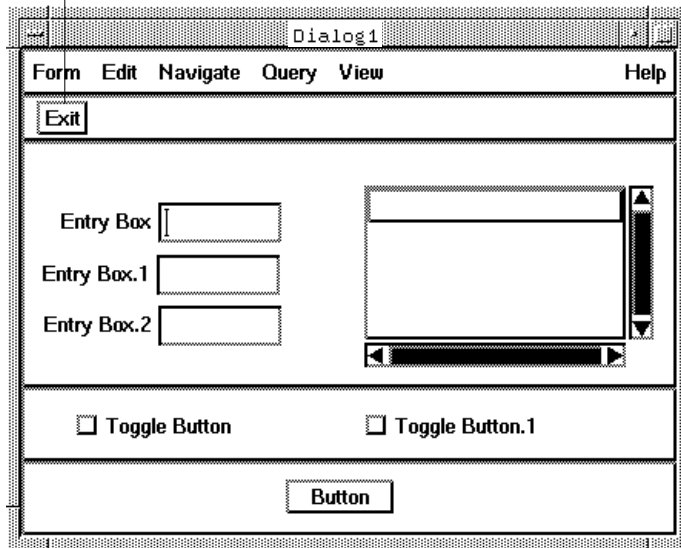
## Adding Menu Options

A menu option is a menu item that, when chosen, calls a macro to perform some action. Menu options can immediately perform the specified action, or they can call dialog boxes that ask for more information before performing the action.

You can also add a menu option to the ExpressLine to make it more easily accessible. For example, to easily exit the dialog box place the Form → Exit option on the ExpressLine:

From the Designer dialog box:

1. Click on **Menu Bar**.
2. Choose **Form** → **Exit**.
3. Click on **Display** in **ExpressLine**.
4. Click on **Abbreviation**.
5. Type **Exit** in the entry area.
6. Click on **OK**.



If the actions you want performed can be logically grouped under one menu name, you may want to add a *cascading* menu. See the next section, "Adding Cascading Menus" for details.

To add a menu option to a menu:

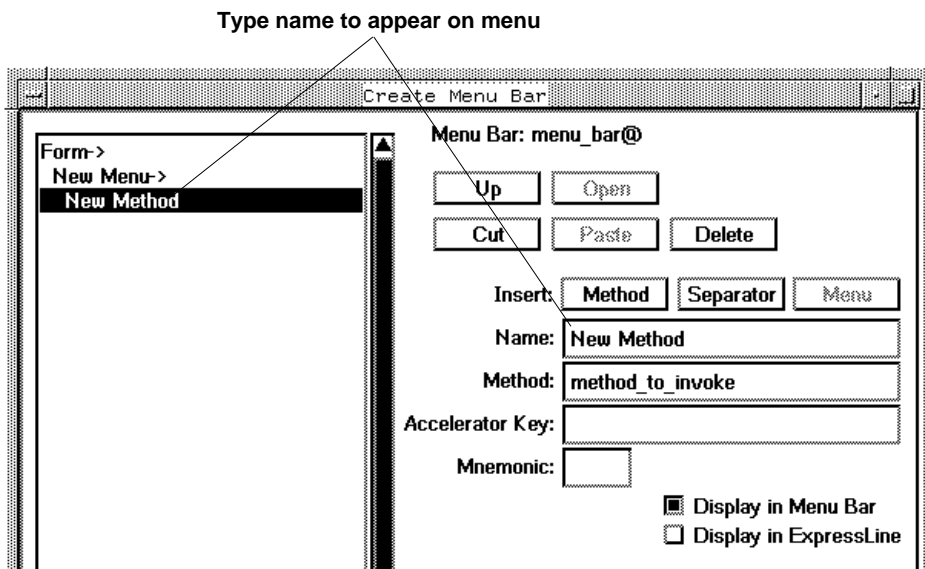
1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.

3. Select the appropriate menu in the list area. Double-click on the menu to quickly update the list area.
4. Select the item in the list area that will appear *after* the menu option you want to add.

If you want the new menu option to be the last option in the list, do not select anything in the list area.

5. Click on **Method**.

When you click on Method, the name New\_Method appears in the list area.



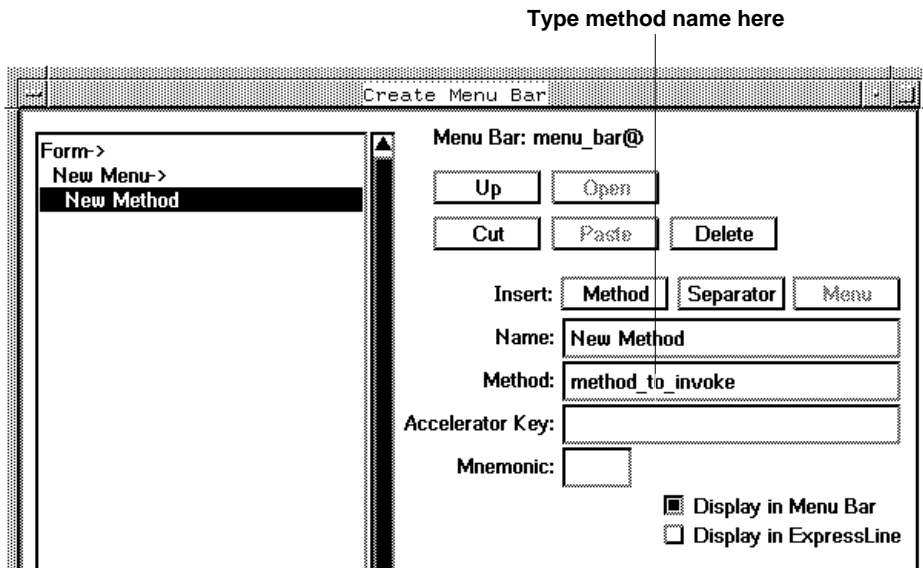
7. Type a name for the menu option in the Name entry area.

To avoid confusion, the name should be unique to the application. The name should be descriptive enough so that you can easily identify what the menu option does. Names can include spaces.

If the menu option you create calls a dialog box, you may want to follow the standard whereby an ellipse (...) at the end of the option name indicates that choosing the option displays a dialog box. For example, if your menu option "Special Save" calls a dialog box, you can type the name as Special Save....

8. In the Method entry area, type (or overtype) the method to be called by this menu option.

The method need not exist to enter it here, however, before the menu option can be used, the method, with the name exactly as it appears here, must exist.



If you want, type an accelerator key for the menu option in the Accelerator key entry area. Use ! for SHIFT, ^ for CONTROL, and

\* for ALT or Meta. If you specify an existing accelerator key, the Menu Bar Editor accepts any other changes you've made on this dialog box, but does not overwrite the existing accelerator key assignment. A message displays indicating that the key is already used. Assign this menu option a new key.

9. You can decide to put this menu option on the ExpressLine in any of three forms: Name, Bitmap, or Abbreviation. Click on the Display in ExpressLine option, then complete the rest of the appropriate entry areas as necessary. You can always add these features to this menu option at any time.

## Adding Cascading Menus

Cascading menus let you group several menu options onto a single menu name. This technique helps keep your pull-down menus brief so you do not need to be bothered with certain types of actions that only have meaning in specific contexts.

In some ways, all pull-down menus act like cascading menus, except that pull-downs appear off the top-level Menu Bars. Cascading menus appear as second-level menus within Menu Bar options. A menu option that displays a cascading menu is indicated by a right pointing arrow beside it. In Applixware, you can use only two levels of menus, that is, a cascading menu from a menu.

To add a cascading menu to a menu, you follow the same steps described earlier for "Adding a Menu" except your starting place is a pull-down menu instead of a top-level menu:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.
3. Be sure the appropriate menu is selected in the list area.

4. Select the menu in the list area that will appear *after* the cascading menu you want to add.

If you want the new cascading menu to be the last menu item in the list, do not select anything in the list area.

5. Click on **Menu**.
6. Type the name for the cascading menu over the New\_Menu in the entry area.
7. Click on **OK** or **Apply**.
8. Double-click on the newly-added cascading menu to display it in the list area.
9. Now, with the new cascading menu displayed, follow the steps described earlier for “Adding Menu Options” to insert the new menu options you want to see on this cascading menu.

## Adding Menu Separators

A menu separator is a horizontal line that appears on a pull-down or cascading menu. It is used as a visual aid to separate menu options so that similar options are grouped together.

In the ExpressLine, a menu separator displays as a blank space. Use it in the same way you use the line in menus—to group items visually.

To add a menu separator:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.
3. Select the menu that will contain the separator. You cannot add a separator to the Menu Bar itself.

4. Select the item in the list area that will be *below* the menu separator you want to add.
5. Click on **Separator**.

When you select Separator, a horizontal line is placed in the menu directly above the menu option that was selected when the insertion was made.

Menu separators do not have names, macros, or accelerator keys, so those entry areas are grayed.

6. By default Display in Menu Bar is turned on for the newly-added separator. You can:
  - Put the separator on only the ExpressLine by turning off the Display in Menu Bar and turning on Display in ExpressLine.
  - Put the separator on both lines by turning on both Display in Menu Bar and Display in ExpressLine.

## Deleting Menu Items

To delete a menu item:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.
3. Select the item to remove from the list area.

The Cut and Delete buttons ungrey, indicating that they can be executed.

4. Click on either **Cut** or **Delete** to remove the menu item.

If you choose Cut, the item is placed in the clipboard and can be retrieved using the Paste button. The menu material remains on

the clipboard until you exit the Create Menu Bar dialog box. If you choose Delete, the item is deleted and is not recoverable.

When a menu item is cut or deleted, the menu option display is automatically updated to show that the item has been removed. The menus themselves are updated when you click on OK or Apply.

If you delete a menu that contains other menu options, those options are also deleted. Similarly, you can cut an entire menu along with its options and paste them into another menu or application.

## Moving Menu Items

You can use the Cut and Paste buttons to move a menu, menu option, or separator to a different location within the Menu Bar of an application.

To move a menu item within an application:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.
3. Select the menu item to move from the list area.
4. Click on **Cut**.

The menu item is removed from the display and placed in the clipboard.

5. Display the appropriate menu in the list area and select the menu item that will be below the place where you want to paste the clipboard item.

If no items are selected, then the clipboard item will be the last item in the menu.

6. Click on **Paste** to place the menu item in its new location.

## Different Menu Bar and ExpressLine Positions

By default, the order in which menu options are presented in the Menu Bar is the same order they are presented in the ExpressLine. You can change that relationship by positioning one version of the menu option on the Menu Bar in its appropriate place and turning off Display in ExpressLine. Then, Cut, Cancel, and Paste a second version of the menu option in its appropriate place with Display in ExpressLine turned on and Display in Menu Bar turned off.

## Exiting the Menu Bar Editor

When you exit the Menu Bar Editor, you can choose to accept all the changes you have made to the Menu Bar, or you can exit without saving any of the changes.

## Exiting and Accepting Menu Bar Changes

To exit the Menu Bar Editor and save all changes that you have made during the editing session, click OK or Apply.

Items which you add to a Menu Bar appear immediately in the current application's Menu Bar when the Menu Bar Editor is exited, with the following exceptions:

- If you add a menu, but do not add any items to the menu, the menu will not appear in the application's menu. A menu must include menu options in order to be displayed.
- If you add a menu option and do not turn on either Display in Menu Bar or Display in ExpressLine for the option, the menu option will not appear in the menu.

For most operations in the Menu Bar Editor, OK is the default button. Any time you press the RETURN key while OK is the default button, you exit the Menu Bar Editor and save any changes you have made to the menu.

## Exiting Without Saving Menu Bar Changes

To exit the Menu Bar Editor without saving any of the changes you have made during the editing session, click Cancel.

The Menu Bar remains in the state it was in prior to opening the Menu Bar Editor.

## Default Menu Bar

A default menu bar is available for application dialog boxes. You can load the default menu bar and modify it instead of creating a menu bar from scratch. To load the default menu bar:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.
3. Click on **Default** in the Create Menu Bar dialog box.

The default menu items appear in the list area.

## Editing Menu Bar Methods

A menu bar contains the methods assigned with the Menu Bar Editor. When you choose a menu bar item, the associated menu bar method is called. To edit menu bar methods:

1. Click on a dialog box name in the Dialog Boxes list area.
2. Click on **Menu Bar**.

3. Click on **Source** in the Create Menu Bar dialog box.

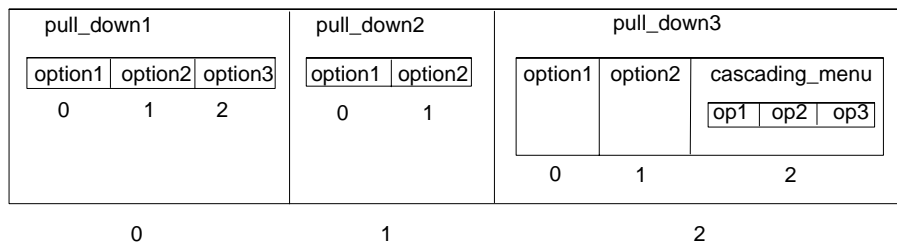
An Edit Source window appears. See Chapter 3 in the *Applix Builder Programmer's Reference*, "Editing Object Methods", for information about using and editing methods.

### Creating a Menu Bar Using an Array

You can define a menu bar using array definitions rather than using the Menu Bar Editor. You can include these array definitions in your dialog box source or you can create a separate macro for the definitions so they are easily identified.

A menu consists of the menu bar, the pull-down menus on the menu bar, and the individual options contained on the pull-down menus. If you want, you can also include cascading menu items in a pull-down menu. A menu bar definition is an array of arrays. The top-level array is the menu bar. The elements in this top-level array are arrays representing the pull-down menus. The pull-down menus consist of arrays that define the options in the pull-down menu. The structure of a menu bar array is illustrated in the following figure.

menu\_bar



**Figure 4-3 Menu Bar Array**

The following sections describe each of the array elements that comprise a menu bar definition. An example of a menu bar definition is also provided.

### Adding a Menu Definition to a Dialog Box

When you define a menu bar with an array you must define the menu bar in the dialog box source. Define a menu bar in the dialog box `initialize_event`. A dialog box `initialize_event` with a menu bar definition has the following format:

```
include "menubar_.am"
set initialize_event
    var object menu
    var menu_bar_definition
    IF NOT object_exists@(this.menu_bar)
    {
        ' menu option definitions
        ' assign definitions to menu_bar_definition
        menu = object_create@("MenuBarClass", "abc")
        menu.data@ = menu_bar_definition
        this.add_child@ = menu
        this.menu_bar@ = menu
    }
endset
```

In the dialog box `init` event you define the menu options, as described later in this chapter. After you define the menu options, you create a `MenuBarClass` object and assign the menu definition to the object with the `set` method `data@`. After assigning the data, add the menu object as a child of the dialog box with the `set` method `add_child@`, then

define the menu object as the dialog box menu bar with the set method `menu_bar@`.

If you define a menu bar in a dialog box `initialize_event` and you do not want it to remain a child of the dialog box you can delete it in the dialog box `terminate_event`. A dialog box `terminate_event` deleting a menu bar definition has the following format:

```
set terminate_event
    var object menu
    menu = this.menu_bar@
    this.delete_child@(menu)
    object_destroy@(menu)
    this.menu_bar@ = NULL
endset
```

The `terminate_event` removes the menu bar as a child of the dialog box, destroys the menu bar object, and sets the dialog box menu bar to `NULL`.

The header file `menubar.am` contains the defined menu bar values and formats. Include this line in the object source if you are using defined values to set menu bar values.

See Chapter 2, "Object Oriented Concepts", for information on `initialize_event`, set methods, and object macros.

## Changing an Existing Menu Bar Definition

If a `MenuBarClass` object exists as a child of a dialog box you can edit the object's source to change the menu bar definition. Change the menu bar definition in the menu bar's `initialize_event`. A menu bar `initialize_event` with a menu bar definition has the following format:

```
set initialize_event
    var menu_bar_definition
    ' menu option definitions
    ' assign definitions to menu_bar_definition
    this.data@ = menu_bar_definition
endset
```

In the dialog box `initialize_event` you define the menu options, as described later in this chapter. After you define the menu options, use the set method `data@` to assign the menu definition to the menu bar.

## Menu Option Definitions

Menu option definitions are arrays that can have up to four array elements:

```
option[x] = "name", "method", "accelerator key", key access
only
```

The array elements are:

name	The name that is to appear for the option in the pull-down menu.  A name preceded by @, in the form @name, is a get method. Use a get method to display a variable string in the menu, such as the state of a toggle button. The get method must be in the menu bar object's source, and the method must return a string.
method	The method that the option calls. Note, that choosing the menu item does not call the method; you must provide a statement in your dialog box methods to call the desired method.

accelerator key	The accelerator key for the method. Use the standard accelerator key notation. For example, to specify the accelerator key CTRL-B, you would supply the argument “^”.
key access only	Indicates whether the option should appear on the menu bar or whether it should only be accessible using the accelerator key. If you specify TRUE, the option is only accessible using the accelerator key. If you specify FALSE, the option appears on the menu bar. The default is FALSE.

The following are examples of menu option definitions:

```
option[0] = "Delete", "delete_it", "^D"  
option[1] = "Move", "move_it", "^M", TRUE  
option[2] = "Uppercase", "make_upper"  
option[3] = "Lowercase", "make_lower"
```

## Pull-down Menu Definitions

Pull-down menus are defined as follows:

```
pull_down = "name", option_array
```

where name is the name that is to appear on the menu bar.

option\_array is the name of the array containing the pull-down menu option definitions.

The following are examples of menu option definitions:

```
pull_down1 = "Operations", operation  
pull_down2 = "Updates", update
```

```
pull_down3 = "Utilities", utils
```

## Cascading Menu Definitions

A cascading menu is defined as follows:

```
option[x] = "name", cascade_option_array
```

name is the cascading menu option name that is to appear in the pull-down menu. cascade\_option\_array is the array that contains the cascading menu option definitions. Cascading menu options are defined in the same way as pull-down menu options.

For example, the third option (options[2]) in this pull-down menu is a cascading menu option:

```
options[0] = "Add account", "add"  
options[1] = "New window", "new_win"  
    type[0] = "Local", "local_map"  
    type[1] = "National", "nat_map"  
    type[2] = "International", "intl_map"  
options[2] = "Maps", type  
options[3] = "Graphs", "graph"
```

## Menu Bar Assignments

You assign pull-down menus to a menu bar using the following notation:

```
menu_bar[x] = pull_down
```

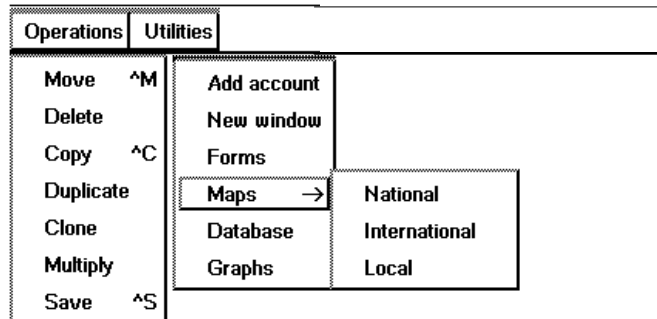
menu\_bar is the name of the menu bar in which you want to place the pull-down menus. pull\_down is the name of the array containing the pull\_down menu definitions.

The following are examples of statements that assign pull-down menus to a menu bar:

```
my_menu[0] = pull_down1
my_menu[1] = pull_down2
my_menu[2] = pull_down3
```

### Menu Bar Definition Example

The following figure shows an example menu bar that contains two pull-down menus. One of the pull-down menus includes a cascading menu.



**Figure 4-4 Menu Bar**

This menu is defined as follows:

```
VAR menu_bar, operate, options1, options2, type, utility
```

```
'Add operations pull-down menu

options1[0] = "Move", "move_it", "^M",FALSE
options1[1] = "Delete", "delete_it"
options1[2] = "Copy", "copy_it", "^C",FALSE
options1[3] = "Duplicate", "dup_it"
options1[4] = "Clone", "clone_it"
options1[5] = "Multiply", "mutiply_it"
options1[6] = "Save", "save_it", "^S",FALSE
operate = "Operations", options1
menu_bar[0] = operate

'Add utilities pull-down menu

options2[0] = "Add account", "add"
options2[1] = "New window", "new_win"
options2[2] = "Dboxes", "dbox"
'add options to cascading menu
    type[0] = "Local", "local_map"
    type[1] = "National", "nat_map"
    type[2] = "International", "intl_map"
options2[3] = "Maps", type    'cascading menu
options2[4] = "Database", "data_base"
options2[5] = "Graphs", "graph"

utility = "Utilities", options2
menu_bar[1] = utility
```

## Saving the Menu in an ELF Data File

If you have created a separate method for the menu bar definition, you need to save the definition to an ELF data file using the `WRITE_DATA_FILE@` macro. The format for the macro is:

```
WRITE_DATA_FILE@(filename, array_name)
```

The `filename`, a string, is the full path name of the file you want to write to. If `filename` is the name of an existing file, the contents of the file will be overwritten.

The `array_name` is the name of the array that you want to write to the data file. For menu bar definitions, `array_name` is the name of the menu bar array.

For example, the following method defines a menu bar having one pull-down menu and writes the definition to the file named `"/user/joe/menus/utility.mb."`

```
set MyMenu
  VAR menu_bar, utility, options, type
  options[0] = "Add account", "add"
  options[1] = "New window", "new_win"
  options[2] = "Dboxes", "dbox"
  'add options to cascading menu
  type[0] = "Local", "local_map"
  type[1] = "National", "nat_map"
  type[2] = "International", "intl_map"
  options[3] = "Maps", type
  options[4] = "Database", "data_base"
  options[5] = "Graphs", "graph"
  utility = "Utilities", options
```

```
        menu_bar[0] = utility

WRITE_DATA_FILE @("/user/joe/menus/utility.mb", menu_bar)
endset
```

## An Example Menu Bar Defined Using Arrays

The following is an example of a menu bar that includes two pull-down menus. The menu bar is defined in a method separate from the dialog box macro.

```
set MakeBar
    VAR bar, menu, file, opts

    menu[0] = "Macro", "my_macro"
    menu[1] = "OK", "ok"
    menu[2] = "Not OK", "not_ok"
    file = "File", menu
    bar[0] = file

    menu[0] = "Option1", "opt1"
    menu[1] = "Option2", "opt2"
    menu[2] = "Option3", "opt3"
    menu[3] = "Option4", "opt4"
    opts = "Options", menu
    bar[1] = opts

    WRITE_DATA_FILE @("/user/joe/my_menubar", bar)
endset

set initialize_event

    this.data@ =
READ_DATA_FILE @("/user/joe/my_menubar")
endset
```

## Making Menu Options Grayed or Toggled

Write a get method `menu_state` to set menu options and states. The method must be in the menu bar object's source. The `menu_state` method for the default menu bar is:

```
get menu_state(func)
    var object dataset,dbox
    dbox = this.parent
    dataset = dbox.data_set@
    case of lowercase@(func)
    case "print"
        return(MENUSTAT#DIMMED)
    case "toggle_view_expressline"
        return(dbox.expressline_status)
    case "toggle_enable_editing"
        if dataset.is_editing_enabled@
            return(MENUSTAT#TOGGLE_ON)
        else
            return(MENUSTAT#TOGGLE_OFF)
    case "allow_dups"
        if dataset.duplicates_info@
            return(MENUSTAT#NORMAL)
        else
            return(MENUSTAT#DIMMED)
    case "goto_first_record", "goto_next_record",
"goto_prev_record",
"goto_last_record"
        if dataset.is_edited@
            return(MENUSTAT#DIMMED)
        else
```

```
        return(MENUSTAT#NORMAL)
    case "insert_record", "delete_record"
        if dataset.is_editing_enabled@ and not
dataset.is_edited@
            return(MENUSTAT#NORMAL)
        else
            return(MENUSTAT#DIMMED)
    case "commit_edits", "cancel_edits"
        if dataset.is_edited@
            return(MENUSTAT#NORMAL)
        else
            return(MENUSTAT#DIMMED)
    endcase
endget
```

The case arguments are the names of methods in the menu bar object's source. When you select a menu item, the menu\_state method is called, then the changed\_event is called.

Use the following defined values to set menu bar option states.

MENUSTAT#NORMAL	Display the menu option normally.
MENUSTAT#DIMMED	Display the menu option grayed.
MENUSTAT#TOGGLE_ON	Display a toggle on to the left of the menu option.
MENUSTAT#TOGGLE_OFF	The menu option is a toggle, currently off.
MENUSTAT#RADIO_OFF	The menu option is a radio, currently off.

MENUSTAT#RADIO_ON	Display a radio on to the left of the menu option.
MENUSTAT#NO_SHOW	Suppress display of the menu option.

The header file `menubar_.am`, located in the `install_dir/axdata/elf` directory, contains the defined menu bar values and formats. Include this line in the object source if you are using defined values to set menu bar values.



---

# 5 Using the Connector Tool

---

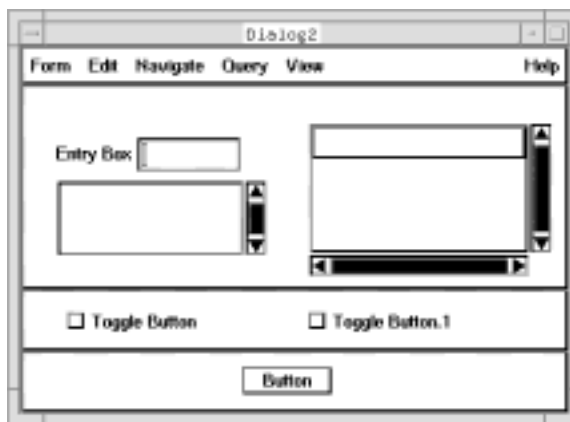
The Connector tool allows you to connect dialog box objects to data sources, as well as modify the attributes and actions of a dialog box object. This chapter covers the following topics:

- Connecting dialog box controls
- Choosing a data source
- Using a display map
- Using a validator
- Connecting a table

## Connecting Dialog Box Controls

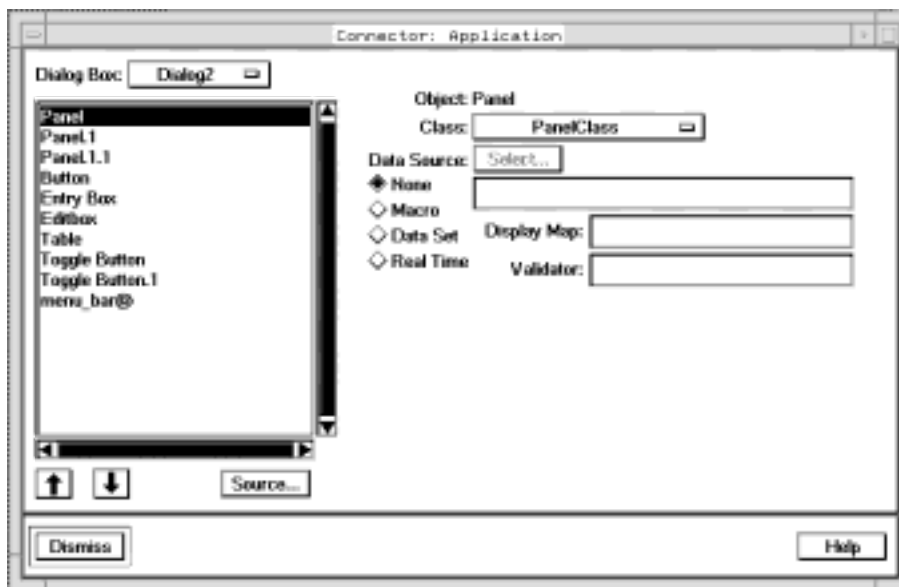
The Connector tool provides a graphical interface for setting dialog box control data sources and attributes. You can edit a control's source to modify its attributes and actions.

The examples in this chapter use the dialog box Dialog2. The dialog box appears below.



To connect dialog box objects:

1. Choose **✳ → Connector** from the Browser window, or click on the Connector icon. The Connector dialog box appears.



2. Choose a dialog box with the Dialog Box option.
3. Click on a dialog box object name in the Dialog Box list area.

The ID of the selected object appears adjacent to the Object label and the inherited class of the objects appears in the Class option. For example, the object Panel in the dialog box Dialog2 has the ID Panel and is a PanelClass object. You can change the inheritance class of the object with the Class option to any of the available classes, even to classes you create. See Chapter 4, "Using the Browser Tool", for information on creating custom classes.

After you choose a dialog box object, you can choose a data source for the object and set display map or validator macros. EditBoxClass, TableClass and CrossTableClass objects have unique connection options. See "Connecting a Table" later in this chapter for information table connection options. See "Connecting an Edit Box" later in this chapter for edit box connection options.

## Choosing a Data Source

Choose a data source for the information you want to display in the control object. You can choose one of the following data source types:

- None
- Macro
- Data Set
- Real Time

### None

Choose the None data source type if you do not want a data source for the control object. For example, a button object does not need a data source. A button object performs an action, it does not directly display information.

### Macro

Choose the Macro data source if you want a macro to retrieve information for the control object. If you choose a Macro data source, type the name of the macro in the Data Source entry area, preceded by @. The @ before the macro name indicates the name is a macro, not a method. For example, you would type the macro example in the Data Source entry area as:

@example

The macro must be accessible by the application. If the macro is not in the object method source, the application searches all other object method sources, then macros in your ELF search path. A macro in the object method source has precedence over all other macros with the same name.

## Data Set

Choose the Data Set data source type if you want data set information to appear in the control object. To connect to a data set after choosing a control object:

1. Choose **Data Set** from the Data Source radio group.
2. Type a data set column name in the Data Source entry area, or click on **Select** to select a data set and column with the Choose Data Set and Column dialog box.

## Typing the Data Set Column

If you are familiar with the names and columns in your data set, you can type the name directly into the Data Source entry area. The data set column format is

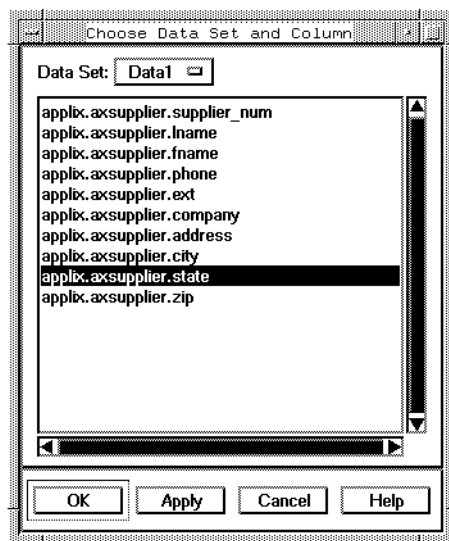
```
data_set->owner.table.column
```

where `data_set` is the name of the data set, `owner` is the owner name of the database table, `table` is the database table name, and `column` is the specific column in the database table.

## Choosing the Data Set Column

Use the Choose Data Set and Column dialog box to choose the data set column you want to connect to the control object. To choose a data set column after choosing a control object and the Data Set data source:

1. Click on **Select**. The Choose Data Set and Column dialog box appears.



2. Choose a data set with the Data Set option.
3. Click on a data set column in the list area.
4. Click on **OK**.

The data set column name appears in the Data Source entry area of the Connector dialog box.

## Real Time

Choose the Real Time data source type if you want Real Time information to appear in the control object. To connect to a Real Time row and field after choosing a control object:

1. Choose **Real Time** from the Data Source radio group.
2. Type a Real Time row and field combination in the Data Source entry area, or click on **Select** to select a Real Time row and field with the Choose Data Feed Row and Field dialog box.

## Typing the Data Feed Row and Field

If you are familiar with the rows and fields in your data feed, you can type the name directly into the Data Source entry area. The data feed format is

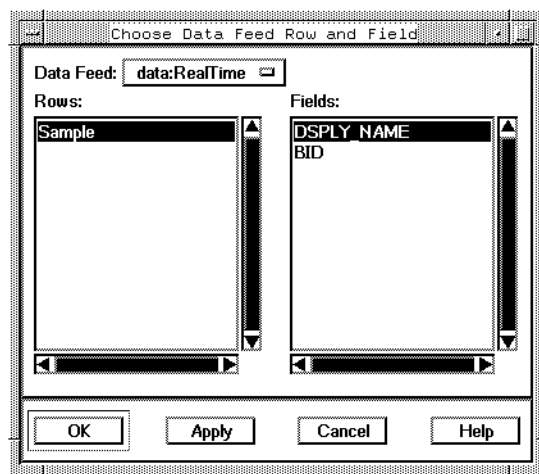
```
data_feed->row^field
```

where `data_feed` is the name of the Real Time data source, `row` is a row in the data feed, and `field` is the specific field in the row.

## Choosing the Data Feed Row and Field

Use the Choose Data Feed Row and Field dialog box to choose the Real Time row and field combination you want to connect to the control object. To choose a row and field combination after choosing a control object and the Real Time data source:

1. Click on **Select**. The Choose Data Feed Row and Field dialog box appears.



2. Choose a data feed with the Data Feed option.
3. Click on a data feed row in the Rows list area.
4. Click on a row field in the Fields list area.
5. Click on **OK**.

The Real Time row and field combination appears in the Data Source entry area of the Connector dialog box.

## Using a Display Map

A display map is a macro or method to manipulate information retrieved from a database or real time data source. You can use a display map to expand, contract, or otherwise change the information displayed in a control object. Use a macro as a display map if the data source value is changed, such as substituting a string for the data

source value. Use a method as a display map if it changes object characteristics, such as changing the color of a control object label. A display map is called when the get method `data_source_value@` is called in your code or by the application.

The information from the data source is passed as an argument to the method or macro. You can use a display map with any control that you can connect to a data set or real time data source. The argument is a single value for most controls. An edit box receives an array of strings as an argument. A table calls the display map for each field in each row. The return type of the information should be in the same form, such as a single value or array of strings, as the input.

The display map tests the information, then returns a value to display in the control object. For example, the following display map macro takes state abbreviations retrieved from the data set. If the retrieved abbreviation is MA, the macro returns Massachusetts, otherwise the macro returns the passed value.

```
macro expand_state(arg)
/* Used as display map macro for this object
*/
  case of arg
  case "MA"
    return("Massachusetts")
  default
    return(arg)
  endcase
endmacro
```

To use a display map, type the name of the method or macro in the Display Map entry area. If you use a display map macro, type the name of the macro in the Display Map entry area, preceded by `@`. The `@` before the macro name indicates the name is a macro, not a method.

For example, you would type the macro `expand_state` in the Display Map entry area as:

```
@expand_state
```

The macro must be accessible by the application. If the macro is not in the object source, the application searches all other object sources, then macros in your ELF search path. A macro in the object source has precedence over all other macros with the same name.

## Using a Validator

A validator is a macro or method to validate information placed into a database. You can use a validator to expand, contract, or otherwise change the information returned to a data set. Use a macro as a validator if the passed value is changed, such as substituting a string for the passed value. Use a method as a validator if it changes the data set object characteristics. A validator is called when the set method `data_source_value@` is called in your code or by the application.

The information from the data source is passed as an argument to the method or macro. You can use a validator with any control that you can connect to a data set or real time data source. The argument is a single value for most controls. An edit box receives an array of strings as an argument. A table calls the validator for each field in each row. The return type of the information should be in the same form, such as a single value or array of strings, as the input.

The validator tests the information, then returns a value to the data set, which updates the value in the data base. For example, the following validator macro takes the state values placed in the object. If a state value is Massachusetts, Connecticut, or Rhode Island, the corresponding abbreviation is returned, otherwise the macro returns the passed value.

```
macro validate_state(value)
/* Used as a validator macro for this object
*/
  case of value
  case "Massachusetts"
    return("MA")
  case "Connecticut"
    return("CT")
  case "Rhode Island"
    return("RI")
  default
    return(value)
  endcase
endmacro
```

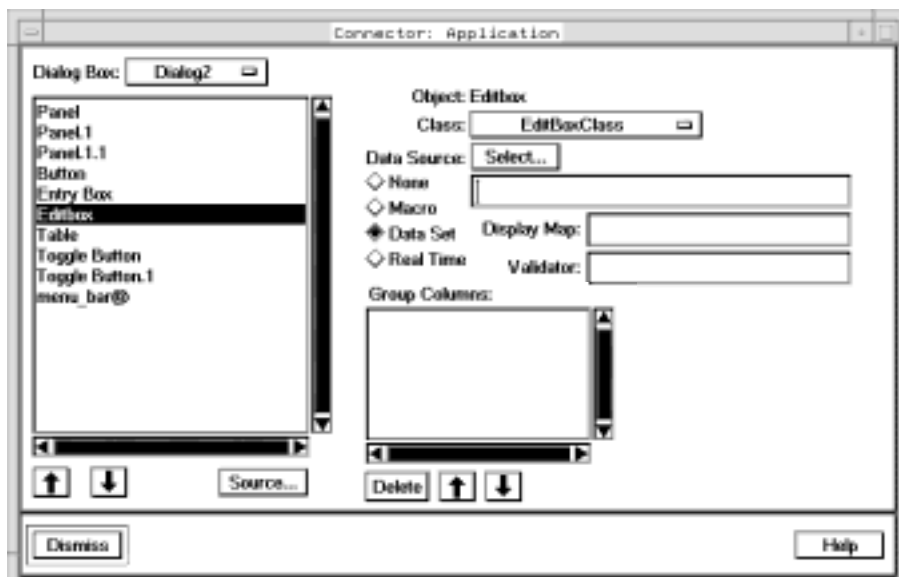
To use a validator, type the name of the method or macro in the Validator entry area. If you use a validator macro, type the name of the macro in the Validator entry area, preceded by @. The @ before the macro name indicates it is a macro, not a method. For example, you would type the macro `validate_state` in the Validator entry area as:

```
@validate_state
```

The macro must be accessible by the application. If the macro is not in the object source, the application searches all other object sources, then macros in your ELF search path. A macro in the object source has precedence over all other macros with the same name.

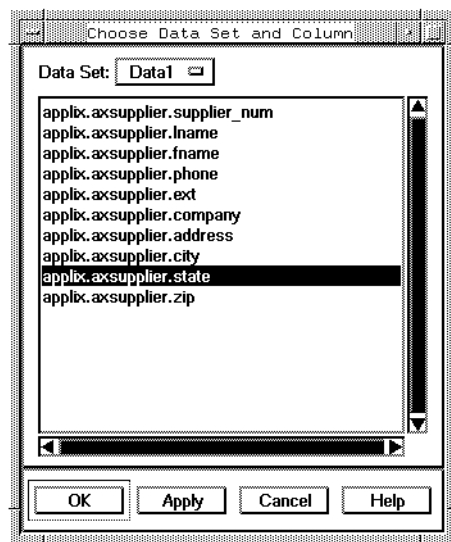
## Connecting an Edit Box

Edit box objects have unique connector options for data sets. An edit box can display multiple columns for a single data set record, or a single Real Time data feed field. When you click on an `EditBoxClass` object in the Dialog Box list area and choose a Data Set data source, the appearance of the Connector dialog box changes.



A Group Columns list area appears. You can add or delete columns in the edit box, in addition to changing the order of the columns in the edit box. To add data set columns to an edit box with a Data Set data source:

1. Click on **Select**. The Choose Data Set and Column dialog box appears.



2. Choose a data set with the Data Set option.
3. Click on a data set column in the list area.
4. Click on **Apply**.
5. Repeat steps 3-5 to add more data set columns to the edit box, then click on **OK** or **Cancel** when you finish adding data set columns.

The added columns appear in the Group Columns list area. To remove a column from the edit box click on a data set column in the list area and click on Delete.

The order of the data set columns in the Group Columns list area determines their display order in the edit box during application execution. Use the Shift Up and Shift Down arrows to change the display order. To change the display order:

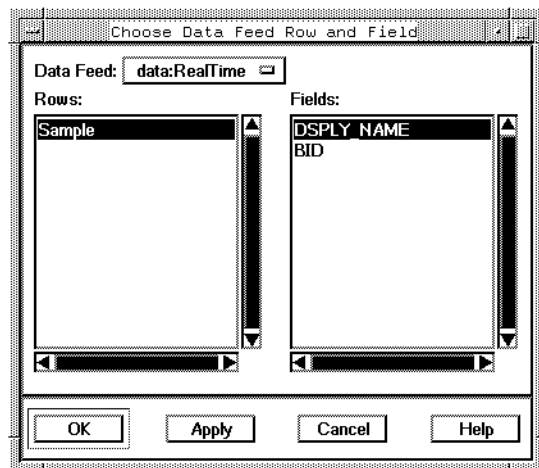
1. Click on a data set column in the Group Columns list area.

2. Click on a Shift Up or Shift Down arrow.

Click on a Shift Up arrow to move the data set column up in the list area and the edit box display. Click on a Shift Down arrow to move the data set column down in the list area and the edit box display.

To add a single Real Time data feed field to the edit box:

1. Click on **Select**. The Choose Data Feed Row and Field dialog box appears.



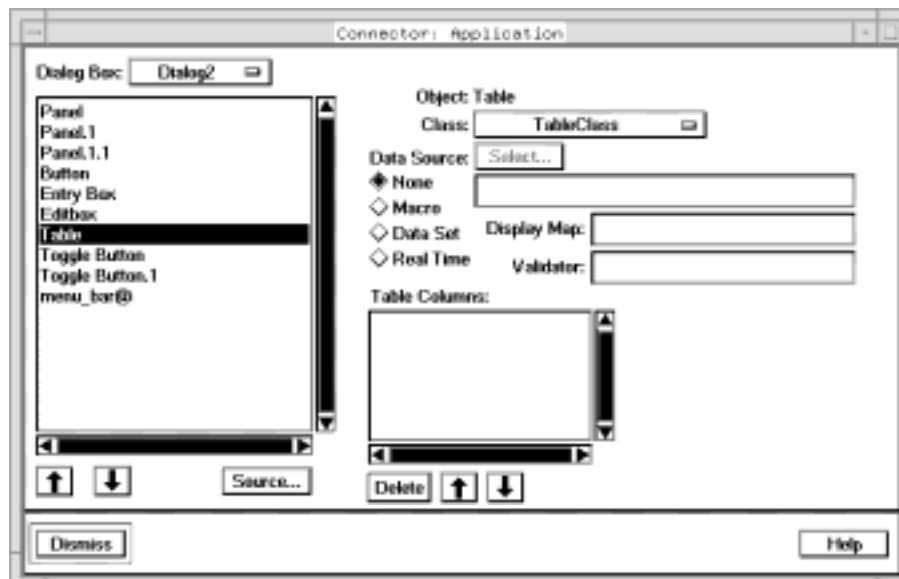
2. Choose a data feed with the Data Feed option.
3. Click on a data feed row in the Rows list area.
4. Click on a row field in the Fields list area.
5. Click on **OK**.

The data feed field is connected to the edit box.

## Connecting a Table

Table objects have unique connector options. Table objects in a dialog box can inherit attributes from either the `TableClass` or `CrossTableClass` classes. Table objects created in the Dialog Box Editor window are `TableClass` objects by default. Use the Class option in the Connector dialog box to change a `TableClass` object to a `CrossTableClass` object.

When you click on a `TableClass` object in the Dialog Box list area, the appearance of the Connector dialog box changes.

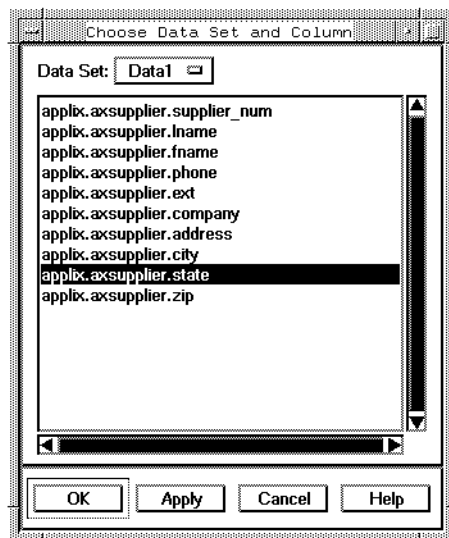


A Table Columns list area appears. A `TableClass` object can have a data set or Real Time data source, while a `CrossTableClass` object can only have a data set data source.

## Creating a Data Set Table

A data set table contains one or more columns from one or more data sets. You can add or delete columns in the data set table, in addition to changing the order of the columns in the table. To create a data set table after choosing a TableClass object:

1. Choose **Data Set** from the Data Source radio group.
2. Click on **Select**. The Choose Data Set and Column dialog box appears.



3. Choose a data set with the Data Set option.
4. Click on a data set column in the list area.
5. Click on **Apply**.

6. Repeat steps 3-5 to add more data set columns to the table, then click on **OK** or **Cancel** when you finish adding data set columns.

The added columns appear in the Table Columns list area. To remove a column from the data table, click on a data set column in the list area and click on Delete.

The order of the data set columns in the Table Columns list area determines their display order in the table during application execution. Use the Shift Up and Shift Down arrows to change the display order. To change the display order:

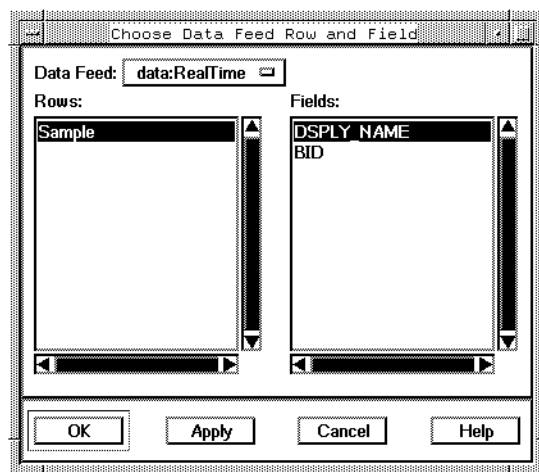
1. Click on a data set column in the Table Columns list area.
2. Click on a Shift Up or Shift Down arrow.

Click on a Shift Up arrow to move the data set column up in the list area, to the left in the table display. Click on a Shift Down arrow to move the data set column down in the list area, to the right in the table display.

## Creating a Real Time Table

A Real Time table contains one or more rows and fields from one or more Real Time data feeds. You can add or delete rows or fields in the Real Time table, in addition to changing the order of the rows or fields in the table. The fields displayed in the table must be valid for all selected records. To create a Real Time table after choosing a TableClass object:

1. Choose **Real Time** from the Data Source radio group.
2. Click on **Select**. The Choose Data Feed Row and Field dialog box appears.



3. Choose a data feed with the Data Feed option.
4. Click on a data feed row in the Rows list area.
5. Drag or CTRL-Click in the Fields list area to select the fields for the table.
6. Click on **Apply**.
7. Repeat steps 3-5 to add more Real Time rows to the table, then click on **OK** or **Cancel** when you finish adding Real Time rows and fields.

The added rows appear in the Table Rows list area and the added fields appear in the Fields list area. To remove a row or field from the table, click on a row or field in the respective list area and click on Delete.

The order of the rows and fields in the list areas determine their display order in the table during application execution. Use the Shift Up and Shift Down arrows to change the display order. To change the display order:

1. Click on a row or field in the respective list area.
2. Click on a Shift Up or Shift Down arrow.

Click on a Shift Up arrow to move the row or field up in the list area. Rows are moved up in table display, fields are moved left in the table display. Click on a Shift Down arrow to move the row or field down in the list area. Rows are moved down in the table display, fields are moved right in the table display.

## Creating a Data Set CrossTable

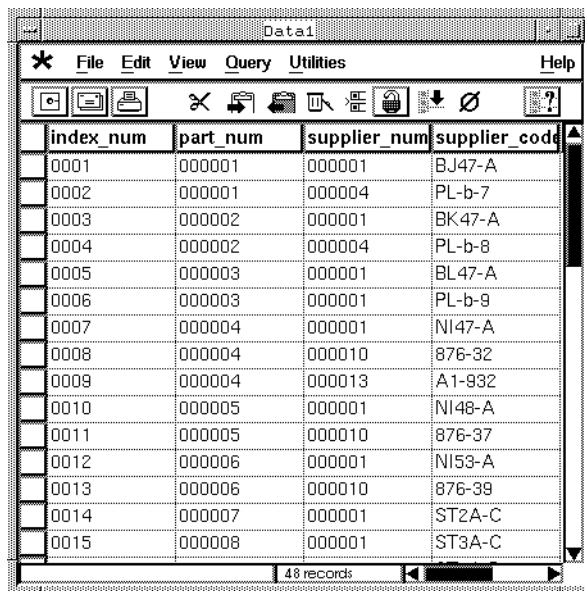
A cross table summarizes database information from two columns in a data set and displays it in a grid format. A cross table presents the requested information in a clear and concise format. Use a cross table to summarize sales, supplier, or other database information instead of using group by and having conditions in the data set. A cross table does not change the information retrieved by a data set.

The values in each data set column become the rows and columns of the cross table. You can choose a row, column, and type of value for a cross table. A value uses an aggregate function, such as count(\*) or avg(), to summarize cross table information.

Use a cross table to group the record information by field. For example, the `axpart_supp` sample table lists part numbers, supplier numbers, and the supplier code for each part number. You can use a cross table to determine the number of times a supplier references a part number.

## Connecting a Table

---



index_num	part_num	supplier_num	supplier_code
0001	000001	000001	BJ47-A
0002	000001	000004	PL-b-7
0003	000002	000001	BK47-A
0004	000002	000004	PL-b-8
0005	000003	000001	BL47-A
0006	000003	000001	PL-b-9
0007	000004	000001	NI47-A
0008	000004	000010	876-32
0009	000004	000013	A1-932
0010	000005	000001	NI48-A
0011	000005	000010	876-37
0012	000006	000001	NI53-A
0013	000006	000010	876-39
0014	000007	000001	ST2A-C
0015	000008	000001	ST3A-C

The `axpart_supp` table contains 48 rows of supplier-part number information. To determine the number of supplier codes a supplier has for each part number, set up a cross table. Set the `supplier_num` column to provide the row heading values, and set the `part_num` column to provide the column heading values. Use the aggregate function `count(*)` to determine the number of times a supplier references a part number.

Cross Table Example

	Part Number						
	000001	000002	000003	000004	000005	000006	000007
000001	1	1	2	1	1	1	1
000002							
000004	1	1					
000005							
000006							
000009							
000010				1	1	1	
000013				1			
000015							

Dismiss

Using a cross table, you can see that supplier 000001 references part number 000003 2 times, while this information is not as obvious using a default query on the database table.

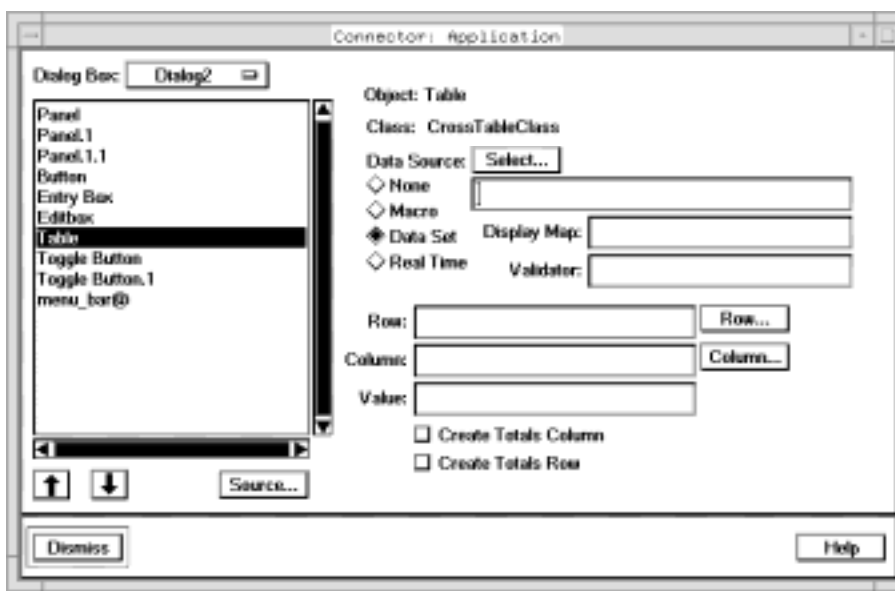
To create a cross table:

1. Choose a TableClass object in the main Browser area.
2. Choose **Object** → **Properties**.
3. Change the class from TableClass to CrossTableClass with the Class option.

To connect a cross table in the connector:

1. Choose a cross table in the Dialog Box list area.
2. Choose **Data Set** from the Data Source radio group.

The appearance of the Connector dialog box changes.



3. Type a data set column name in the Row entry area, or click on Row to choose a data set and column with the Choose Data Set and Column dialog box. The column name in the Row entry area provides the row values for the cross table.
4. Type a data set column name in the Column entry area, or click on Column to choose a data set and column with the Choose Data Set and Column dialog box. The column name in the Column entry area provides the column values for the cross table.
5. Type the value function for the cross table in the Value entry area. An aggregate function, such as count(\*) or avg(), is usually used to summarize cross table information.

6. Turn on the Create Totals Column or Create Totals Row options to create totals of the aggregate values returned by the value function. Create Totals Column creates a column in the table containing the total for each cross table row. Create Totals Row creates a row in the table containing the total for each cross table column.

The cross table rows and columns appear during the application execution.

*Connecting a Table*

---

---

# 6 Using the Debugger Tool

---

This chapter covers the following topics:

- Starting the Debugger
- Selecting objects
- Setting break points
- Setting watch list variables
- Running an application in the Debugger

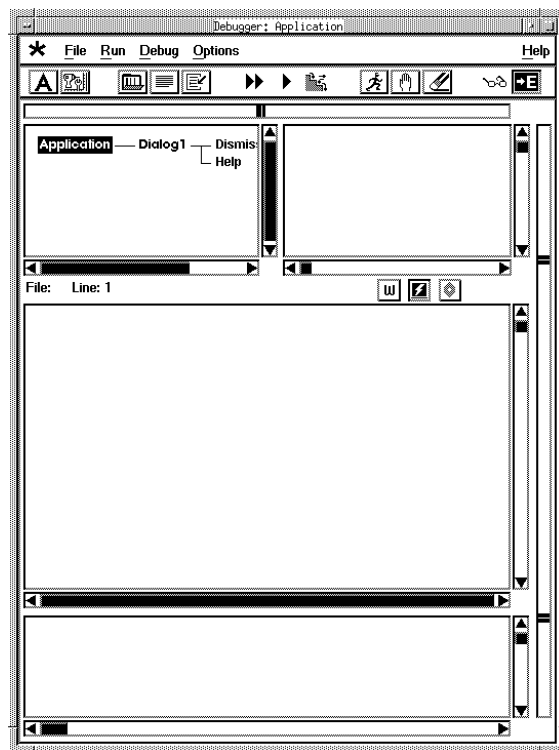
## Starting the Debugger

You can use the Debugger to interactively follow the execution of an application statement by statement. The Debugger discovers any programming logic errors that exist.

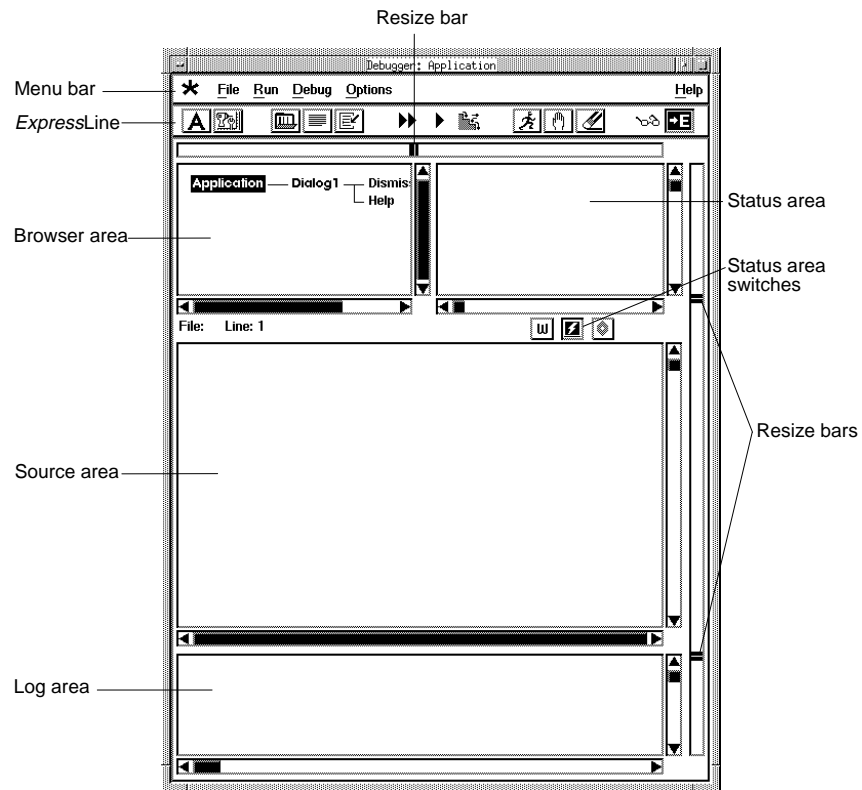


Debugger icon

To start the Debugger tool, choose **★ → Debugger** from the main menu, or click on the Debugger icon. The Debugger window appears, loaded by default with the object method source of the object selected in the Browser window. The Source area of the Debugger window is empty if the selected object does not have executable code in its object method source.



You can resize the Debugger window and change the ratios of the components within the window. The Debugger window consists of the following components:

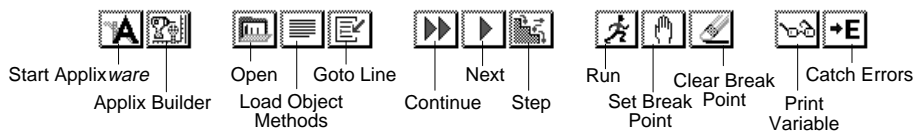


- Browser area**      The Browser area contains the same object hierarchy found in the Browser window. See "Selecting Objects" in the next section for information on selecting objects in the Browser area.
  
- Source area**      The Source area contains the source of the selected object or methods encountered during debugging.

Log area	The Log area displays your current location in the debugging process.
Status area	The Status area is a multi-use display-only area. Use the Status area switches to display the call stack, break points, or watch list variables in the debugging process.
Resize bars	The Resize bars change the size of the Debugger window areas. To change the proportion of an area in the Debugger window, drag the appropriate horizontal or vertical resize bar.

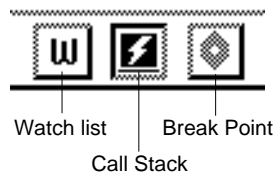
## ExpressLine Icons

You can run an application in the Debugger window after you select application objects and set break points. Use the ExpressLine icons to move through an application in the Debugger window.



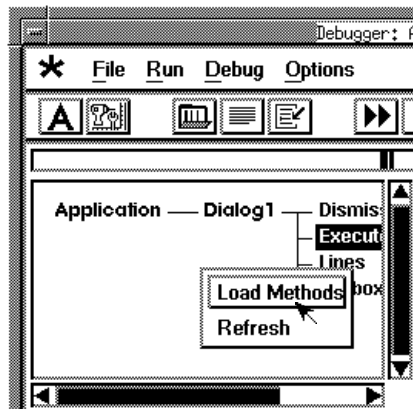
## Status Area Switches

You can use the Status Area switches to change the information displayed in the status area. Use the status area to display the call stack, break points, or watch list variables in the debugging process.



## Popup Menus

Popup menus are available in the Debugger window areas to provide quick access to the various debugging options. When the mouse pointer is in the Browser, Source, Log, or Status area of the Debugger window and you press the right mouse button a popup menu appears. If you choose a menu option and release the mouse button the editing action occurs and the popup menu disappears.



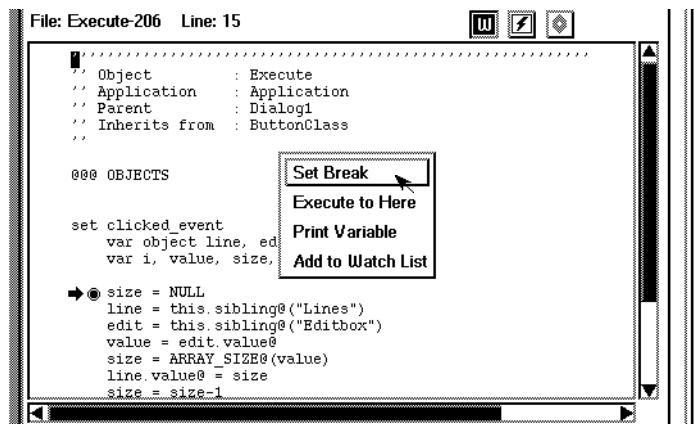
The Browser area popup menu contains options for loading the object method source of the selected object or refreshing the Browser area display.



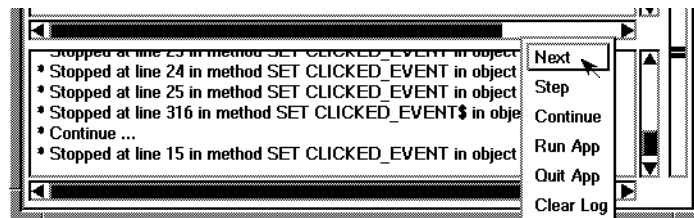
The Status area popup menu options depend on the information displayed. If the Status area is set for watch list variables, the popup menu contains options for removing the selected variable from the watch list, expanding or contracting multiple-level variables, or refreshing the Status area display.

If the Status area is set for call stack, the popup menu contains an option to load the object method source of the selected call stack item into the Source area.

If the Status area is set for break points, the popup menu contains options for clearing the selected break point, clearing all break points, or loading the object method source of the selected break point into the Source area.



The Source area popup menu contains options for setting a break point, executing an application to a selected point in the object method source, printing a selected variable, or adding a selected variable to the watch list.



The Log area popup menu contains options for running the application in the Debugger and clearing the log area.

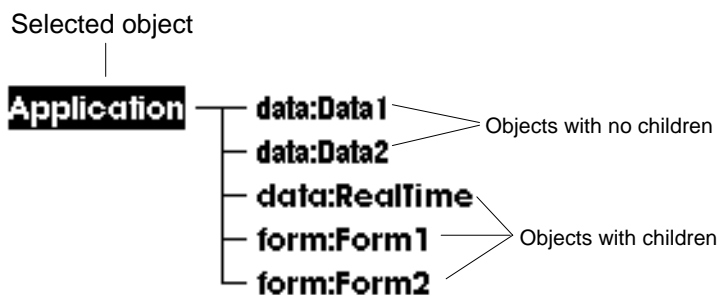
See "Running an Application in the Debugger" later in this chapter for information on setting and using debugging options.

## Selecting Objects

To debug an application object, you must select the object and set break points in the object source. You can set break points in multiple application objects. See "Setting Break Points" later in this chapter for information on setting break points in an object source. To select an application object:

1. Double-click on the ApplicationClass object to expand the object hierarchy.

Object names with a Helvetica Narrow font do not have children. They appear green on color displays. Object names with an Avant Garde font have children. Objects with children also appear brown on color displays. Double-click on an object with children to expand the object hierarchy. Double-click on a parent object to contract the object hierarchy.



2. Select the object to debug. Selected objects are highlighted.
3. Click on the Load Object Methods icon.

The object source appears in the Source area.

## Setting Break Points

Set break points in the object source where you want the debugger to suspend execution of the application.

If your object source calls other methods, you can also set break points in the called methods. If a break point is encountered in a called method, the Source area automatically updates to show the method.

To set a break point:

1. Place the cursor at the location in the object method source where you want to suspend execution.
2. Choose **Debug** → **Set Break Point** or click on the Break Point icon.



Break Point icon

A message appears in the Log area indicating the break point setting.

When the application is run in debugger mode, execution is suspended immediately prior to the statement at which the break point is set.

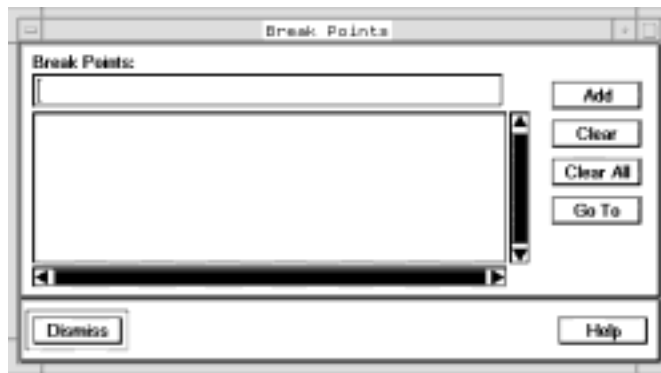
A break point only applies to executable statements. If you set a break point at a non-executable statement, application execution is not suspended at that point.

## Setting Multiple Break Points

You can set multiple break points in an application with the Break Points dialog box. To set multiple break points:

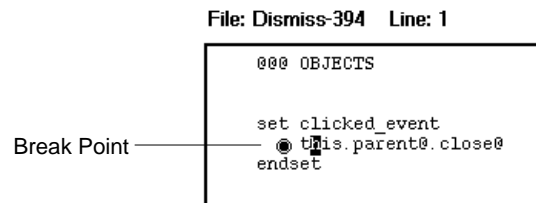
1. Place the cursor at the location in the object method source where you want to suspend execution.

2. Choose **Debug** → **Break Points**. The Break Points dialog box appears.



3. Click on **Add**.

The file name, line number, and object name of the break point appear in the Break Points list area. A message appears in the Log area indicating the break point setting. A break point appears visually in the Source area to the left of the line.



In addition to setting break points, use the Break Points dialog box to clear break points or go to a line with a break point.

To go to a line with a break point, click on a break point in the Break Points list area. The object source containing the break point appears in the Debugger window, and the break point is highlighted.

## Clearing Break Points

To clear a single break point:

1. Place the cursor at the break point location in the object method source.
2. Choose **Debug** → **Clear Break Point**, or click on the Clear icon.



Clear icon

The break point is removed. You can also remove a break point with the Break Points dialog box. Click on the break point in the Break Points list area and click on Clear.

To clear all break points in an application, choose **Debug** → **Clear All Break Points** from the Debugger window, or click on **Clear All** in the Break Points dialog box.

## Setting Watch List Variables

Set watch list variables to observe the values contained in the variable during application. You can see the changes in multiple variables without stopping to individually print each variable.

The Status area must be set to Watch List to observe watch list values. To add a variable to the watch list:

1. Select the variable in the object method source.
2. Choose **Debug** → **Add to Watch List**. The variable is added to the watch list.

## Setting Watch List Variables

---



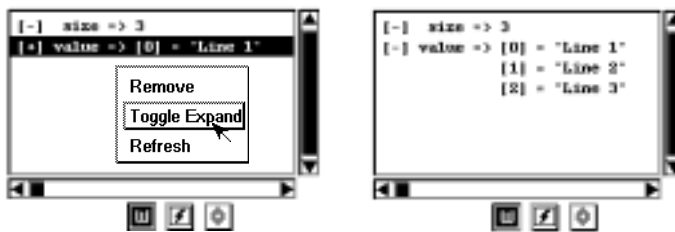
## Using Watch List Variables

The watch list variables display the value of the variable at the current point of execution. A variable that contains an array must be expanded to display all the elements of the array. For example, the following `clicked_event` has a variable value that contains an array of strings, and a variable `size` which is set to the array size.



At the first break point both variables are NULL because they are not set. At the second break point size is 3 and value is an array of three elements. A + symbol in the brackets next to a watch list variable indicate the variable contains multiple values. To expand a variable with multiple values:

1. Select the watch list variable in the Status area.
2. Choose Toggle Expand from the popup menu options. The variable is expanded in the display.

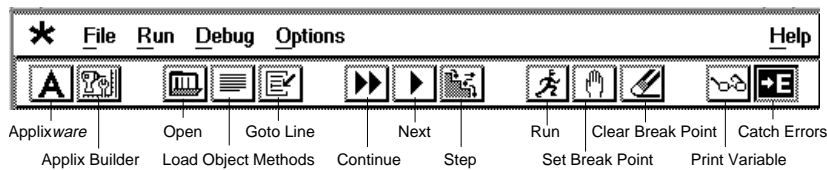


To contract a variable with multiple values:

1. Select the first element of the watch list variable in the Status area.
2. Choose Toggle Expand from the popup menu options. The variable is contracted in the display.

## Running an Application in the Debugger

You can run an application in the Debugger window after you select application objects and set break points. Use the ExpressLine icons to move through an application in the Debugger window.



You can follow the execution of an application from statement to statement by stepping through the object source. You can begin stepping through an application at any break point. To begin debugging choose Run → Run Application. The application executes, stopping at the first break point it encounters.

To step through an application, click on the Step icon while at a break point. When you click on the Step icon, the next executable statement in the application is executed. Click on the Step icon again to execute the next statement, or click on the Continue icon to continue processing of the application without stopping after every executable statement.

If you click on Step while at a statement that calls another method, the Debugger window is automatically updated to show the called method. You can then click on the Step icon to step through the called method. When the called method is completed, the Debugger window is updated to show the calling method.

You can also step through a method using the Next icon. Like Step, the Next icon will execute a single statement and stop. However, if you click on the Next icon while at a statement that calls another method, the called method is executed and processing continues with the next statement in the calling method; Next does not step through a called method.

## Handling Errors

Errors in programming logic may cause an application failure. You can handle errors in a debugging session in two ways:

- Debugger catches errors
- Application handles errors

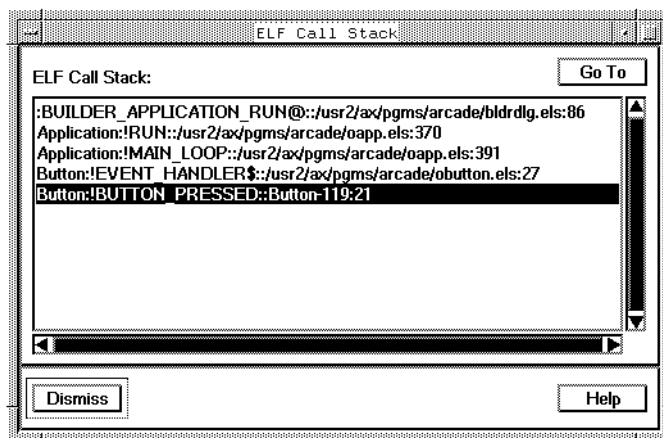
Choose error handling with the Options → Catch Errors option. Turn on the option to have the Debugger window catch errors. The option is on by default. The source containing the error appears in the Debugger window Source area, and the beginning of the line containing the error is highlighted.

Turn off the option to let your application source code handle thrown errors. You can write an error handler using `ErrorClass` methods.

## Moving Through the Call Stack

While you debug, it is often helpful to move up or down the call stack to follow program execution or to set additional break points. For example, while you are following the execution of a called method, you can move up the call stack to the calling method to set additional break points in the calling method or to see the value of variables in the calling method. You can then move back down the call stack to the called method to continue following its execution.

When application execution stops at a break point, choose Debug → Call Stack. The ELF Call Stack dialog box appears.



The object name, object method, object source file name, and line number for all called methods appear in the ELF Call Stack list area. Click on a line in the list area and the source containing the method appears in the Debugger window. The beginning of the line containing the called method is highlighted.

## Displaying Variable Values

While application execution is suspended, you can display the current values of any variables in an object source as follows:

1. Select the variable in the object source.
2. Choose **Debug** → **Print Variable**, or click on the **Print Variable** icon.

The value for the variable is displayed in the Log area. To display values of multiple variables use watch list variables, as described in "Setting Watch List Variables" earlier in this chapter.

## Execute to Here

The Execute to Here popup menu option acts as a temporary break point during the debugging process. You can use the option when the debugging process is stopped at a break point. Programming executions continues from the stopped point until it reaches the point designated by the Execute to Here option. To use the option:

1. Place the cursor at a line in the object method source displayed in the Source area.
2. Press the right mouse button and choose the Execute to Here option from the popup menu.

## Changing Debugging Status

You can change the application debugging status while in debug mode. You can:

- Stop debugging

- Start debugging
- Quit the application

## Stop Debugging

You can stop debugging before, during, or after you start an application in debugger mode. Toggle off the Debug → Enable Debugging menu option. Application execution is suspended if you stop debugging while the application is stopped at a break point. Application execution proceeds if you stop debugging before an application is in debug mode or if the application continues after a break point.

## Start Debugging

Toggle on the Debug → Enable Debugging menu option to enable debugging after you stop debugging. The application execution stops at the next break point it encounters, whether the application is started before or after choosing the Debug → Enable Debugging option.

## Quit the Application

Choose Run → Quit Application to stop application execution and dismiss all application dialog boxes. The Run → Quit Application option does not close the Debugger window or other Applix Builder tools. You can use the Run → Quit Application option to exit an application while keeping the application in debugger mode.

*Running an Application in the Debugger*

---

---

# 7 Running and Distributing Applications

---

Various options are available for distributing and running Applix Builder applications. You can distribute applications in many different formats. This chapter covers the following topics:

- Running applications from Applix Builder
- Using axmain
- Making a macro
- Creating a turbo file
- Creating a distribution file

## **Running Applications from Applix Builder**

You can run macros or the current application from Applix Builder. Choose **★ → Run** from the Applix Builder main menu to run the current application.

Choose **★ → Run Macro** from the Applix Builder main menu to run a macro. The macro runs independently of the current application, so you can run an application macro while working on another Builder application.

See "Making a Macro" later in this chapter for information about creating an application macro.

## **Running Applications from UNIX/Linux**

Once you have written a Builder application, you can run that application from the UNIX/Linux command line using `axmain`.

### **Before You Run `axmain`**

Before you run an application with `axmain`, you must set the appropriate environment variables for your system. The table below shows how to set the environment variables required by `axmain`:

<b>Operating System</b>	<b>Command to Set the Environment Variables</b>
UNIX/Linux	setenv LD_LIBRARY_PATH /installdir/axdata/axshlib (cshell) set env LD_LIBRARY_PATH=/installdir/axdata/axshlib export LD_LIBRARY_PATH (sh/bash/ksh)
Windows 95 and Windows NT	set PATH=c:\axdata\axshlib

## Running a Macro

You can use `axmain` to run a macro from a shell window. Use `-macro` as the first argument with `axmain` to run an installed or ELF built-in macro. If `-macro` or `-run` is not passed as the first argument, `axmain` assumes the passed argument name is a macro and attempts to install and execute it. Type the macro name with its `.am` macro file extension. If the macro is not in the current shell window directory, type the full path with the macro name.

For example, for the macro `bld.am` in the `/user/applix/axhome/macros` directory, and Applix Builder installed in the `/apps` directory, type the following command in a shell window:

```
/apps/axdata/axmain /user/applix/axhome/macros/bld.am
```

After the macro completes execution the `axmain` process remains. Type `CTRL-C` in the shell window to terminate the process and regain shell window control.

You can also use the Applix Builder Run resource to load and run macros. See "Running Multiple Applications" later in this chapter for more information on running macros.

## **Running an Application**

You can use `axmain` to run an application from a shell window. Use `-run` as the first argument with `axmain` to run an application. If `-macro` or `-run` is not passed as the first argument, `axmain` assumes the passed argument name is a macro and attempts to execute it. Type the application name with its `.ab` or `.abo` application file extension. Applix Builder files have a `.ab` file extension. Applix Builder executable files have a `.abo` file extension. If the application is not in the current shell window directory type the full path with the application name.

For example, for the application `sample.ab` in the `/user/applix` directory, and Applix Builder installed in the `apps` directory, type the following command in a shell window:

```
/apps/axdata/axmain -run /user/applix/sample.ab
```

After the application completes execution the `axmain` process remains. Type `CTRL-C` in the shell window to terminate the process and regain shell window control.

You can also use the Applix Builder Run resource to run applications. See "Running Multiple Applications" later in this chapter for more information on running one or more applications.

## **Running Multiple Applications**

Use the Applix Builder Run resource to run multiple applications and macros against a single `axmain` process. Each application or macro becomes a task in the `axmain` process.

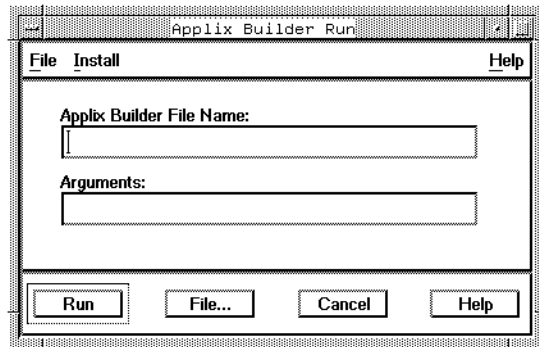
The Applix Builder Run resource also returns control of the shell window when it is dismissed. The `axmain` process is terminated when the resource is dismissed.

The Applix Builder Run resource file is located in the `/install_dir/axdata/elflib/eng` directory, where `install_dir` is the directory where you installed Applix Builder. To run the Applix Builder Run resource, type `-run` as the first argument to `axmain`, then type `runtime.abo` with its full path as the second argument.

For example, for Applix Builder installed in the `apps` directory, type the following command in a shell window:

```
/apps/axdata/axmain -run /apps/axdata/elflib/eng/runtime.abo
```

The Applix Builder Run dialog box appears. Use the dialog box options to run multiple applications or macros against a single `axmain` process.



## Running an Application

To run an application from the Applix Builder Run dialog box:

1. Choose **File** → **Open**.
2. Choose the file to open with the Open dialog box options, then click on **Open**.

The file name appears in the Applix Builder File Name entry area. You can also double-click in the entry area and type the full path

name of the file, with the appropriate file .ab or .abo application file extension.

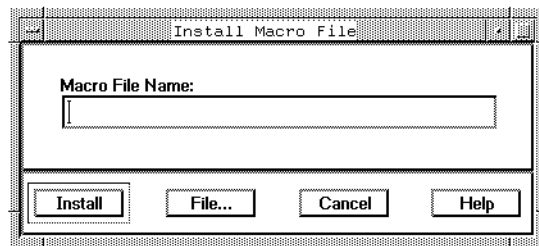
3. Type any arguments for the application the Arguments entry area.
4. Choose **File** → **Run Application**.

The application begins execution. Repeat the steps to run additional applications.

## Running a Macro

You can run ELF macros created with Applix Builder or other Applixware applications with the Applix Builder Run dialog box. You must install a macro before running it. To install a macro:

1. Choose **Install** → **Macro File**. The Install Macro File dialog box appears.



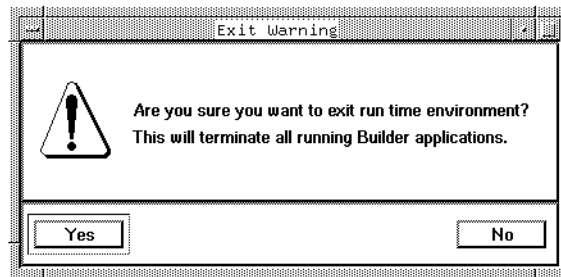
2. Click on **File**.
3. Choose the file to install.
4. Click on **Install**.

After you install the file choose **File** → **Run Macro** or press the F8 key. The Run Macro dialog box appears. Type the macro name in the entry area. Since the file is installed you do not have to type the full path name or .am file extension.

You can use the Files option in the Run Macro dialog box to run macro files created with Applix Builder. You can run macros files created with Applix Builder without installing them, but only with the Files option. When you choose the Files option the Files dialog box appears. Use the dialog box options to choose the Applix Builder created macro file.

## **Exiting axmain**

Choose File → Exit to exit the Applix Builder Run dialog box and all applications and macros. Before exiting the Exit Warning dialog box appears.



This provides a graceful way to exit applications. You have the opportunity to perform a final save of information, if your application allows, or a chance to allow an application to finish execution. Click on Yes to exit the Applix Builder Run dialog box and terminate the axmain process. All applications are terminated. Click on No to dismiss the Exit Warning dialog box and continue application execution.

## Distributing Applications

Use the main Browser window Tools menu options to distribute an application in various formats. The application format depends on how you want to use the application. You can distribute an application as:

- a macro
- a turbo form application
- a distribution form application

This section discusses creating the different application formats for distribution. See "Running an Application" earlier in this chapter for information on running an application.

### Making a Macro

An application created as a macro will run from Applix Builder or Applixware as a macro. Running an application as a macro allows you to run an application with `axmain` without opening it in Applix Builder. You can run one application macro while you are working on another Builder application. To make a macro from a Builder application:

1. Choose **Tools** → **Make Macro** on the Builder main menu. A Save As dialog box appears.
2. Type a macro name.
3. Change the directory of the saved file, if desired

4. Click on **OK**.

The macro is saved in your `/user/axhome/macros` directory, where `user` is your home directory. The Make Macro option also places an executable copy of the application in your `/user/axhome/macros` directory. The file name of the executable file is the name of the application with a `.abo` file extension, indicating this is an Applix Builder executable file. The macro only runs against this executable file. If you make different macros for an application, only the latest copy of the executable file is saved in your `/user/axhome/macros` directory.

If you distribute the macro for use on other systems, the macro must be distributed with the executable file. You can edit the macro in your system editor to set the path names for the executable file.

## Creating a Turbo File

An application saved in turbo form creates a deployable application that requires less memory and loads fast. You can run a turbo application from Applix Builder or with `axmain`. An application saved in turbo form does not localize external files, the external files must be available to the application for it perform properly. To create a turbo file:

1. Choose **Tools** → **Create Turbo** on the Builder main menu. The Save As dialog box appears.
2. Type a file/pathname.
3. Click on **Save**.

The file is saved in the directory with a `.abo` file extension. The file is ready to run as is, or you can make additions to the application with the Applix Builder tools. Changes and deletions of existing objects in the application are limited because all resources are localized in a turbo distribution.

## Creating a Distribution File

An application saved in a distributable form bundles all object classes and resources with the application, creating an application with resources that can be changed or updated upon the installation machines. You install a distributed application with axmain. To create a distribution file:

1. Choose **Tools** → **Create Distribution** on the Builder main menu. The Create Distribution dialog box appears.
2. Enter a file/pathname.
3. Click on **Save**.

The file is saved in the directory with a .abd file extension, indicating that it is a distribution file. Use the axmain Applix Builder Run resource to install a distributed application. See "Installing a Distribution File" earlier in this chapter for more information.

## Installing a Distribution File

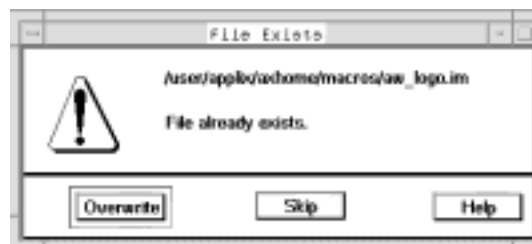
You can install an Applix Builder distribution file with the Applix Builder Run dialog box. The installing procedure places the application, class library, and resource files in a directory on the installing machine. You can change or update any of the files saved in the distribution file. To install a distribution file:

1. Choose **Install** → **Builder Distribution**.
2. Choose the file to install with the Open dialog box options, then click on **Open**. Another Install Builder Distribution dialog box appears.



3. The current directory of the .ab file appears in the Install in directory entry area. Type a different directory path name to install the files in another directory.
4. Choose the installation mode with the Install Mode option.
5. The resource files for the application appear in the External Files List list area. To change the installation path of a resource file, select the file in the list area and type the new path name in the New Directory entry area.
6. Click on **Install**.

The Interactive install mode prompts you for an installation choice for each installation file that conflicts with an existing file. A File Exists dialog box appears for each installation conflict.



Click on Overwrite to overwrite the existing file with the installation file. Click on Skip to leave the existing file and continue the installation procedure.

The files are installed in the directory. See "Creating a Distribution File" later in this chapter for information on creating an Applix Builder distribution file.

---

# 8 Advanced Topics

---

Applix Builder provides advanced capabilities for project development. This chapter covers the following topics:

- External object libraries
- External resources
- C++ class libraries
- Remote objects
- Common dialog boxes
- Applix Builder Bitmaps

## Object Libraries

In addition to using external object classes to decrease duplicate coding, you can use object libraries. An object is a specific instance of a class. For example, you can create a prototypical dialog box, or a Real Time record, or any specific object you want to reuse or share with other applications.

An object library contains one or more objects. The objects in an object library are not necessarily related, the grouping of objects in a library depends on your preferences.

As with external object classes, you can save an object library as a linked object library or a localized object library.

### Creating an Object Library

You can create an object library by saving an object from an application as a library, or by creating a new library. Once an object library exists you can add objects and make other edits to the library.

### Saving an Object as a Library

You can save any object in the application as a library. To save an object as a library:

1. Choose an object in the Browser window.
2. Choose **Object** → **Save As Library**. The Save As dialog box appears.

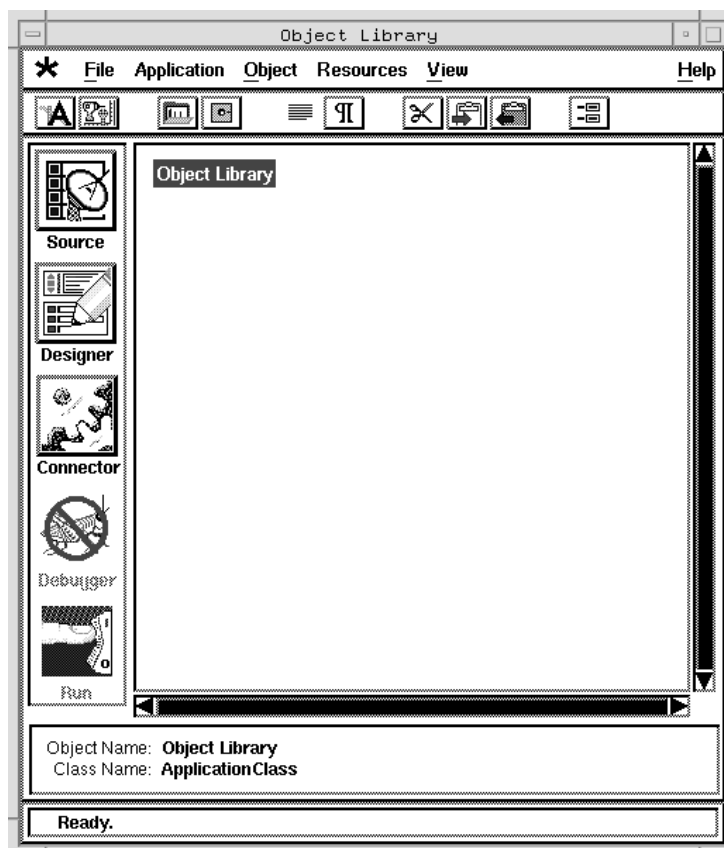
Specify a name, location, and permissions for the object library file. You can save the file with any file extension. However, the Builder default search when loading an object library is for files with a .oll file extension.

3. Click on **Save**.

## Creating a New Library

You can create a new object library with created objects, or objects copied from the current application. To create a new library:

1. Choose **Tools** → **Edit Object Libraries**. The Object Library dialog box appears.



2. Choose **Object** → **Add** to create a new library object, or cut or copy existing objects from the Browser window and paste them in the Object Library dialog box.

See Chapter 2, "Using the Browser Tool", for information on cutting, copying, pasting, and adding objects.

3. Choose File → Save As when you are finished adding objects to the library.

Specify a name, location, and permissions for the object library file. You can save the file with any file extension. However, the Builder default search when loading an object library is for files with a .oll file extension.

4. Click on **Save**.

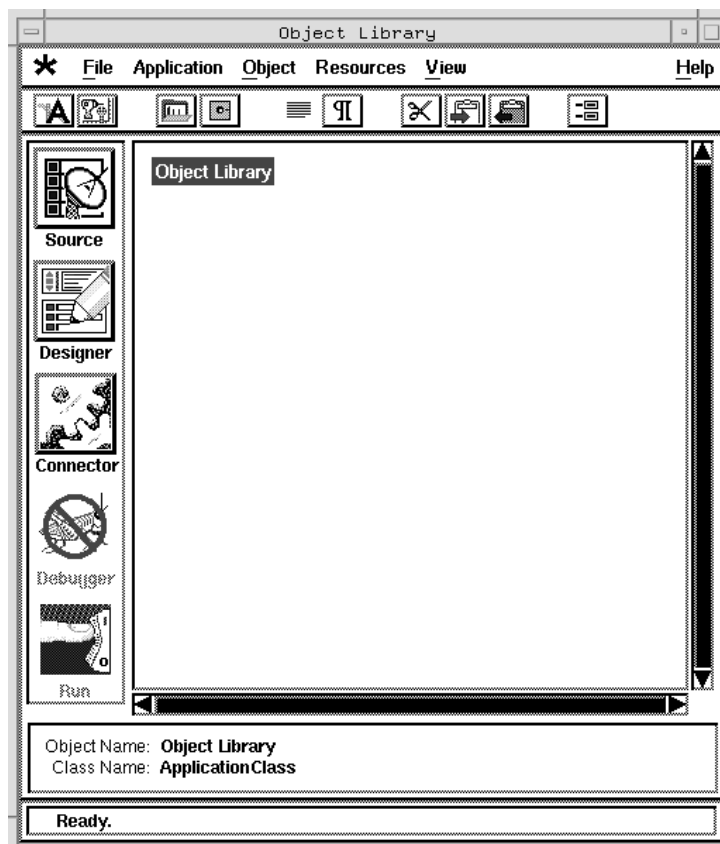
## Editing an Object Library

You can change the contents of an object library to reflect changes in functionality or design. You can edit an object library by adding, deleting, or editing objects in the library, or by merging libraries together.

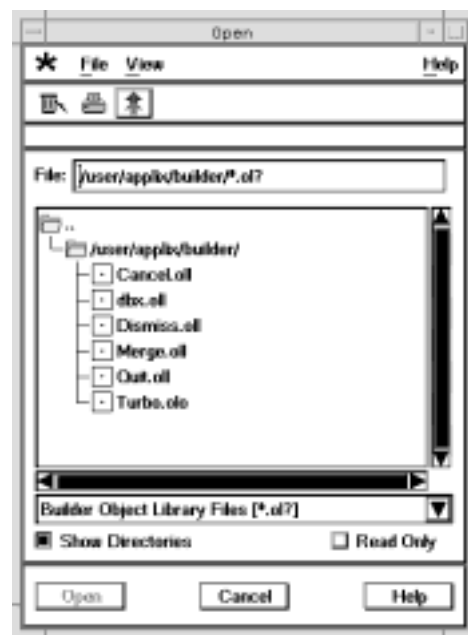
**NOTE:** An application using an external object library does not receive edits from the library until the library is reloaded, or the application is exited and reopened.

To edit an object library:

1. Choose **Tools** → **Edit Object Libraries**. The Object Library dialog box appears.



2. Choose **File** → **Open**. The Open dialog box appears.



Use the dialog box options to search directories and files for the object library. The default search is for files with a .ol file extension. Change the search wildcard if your object libraries have a different file extension.

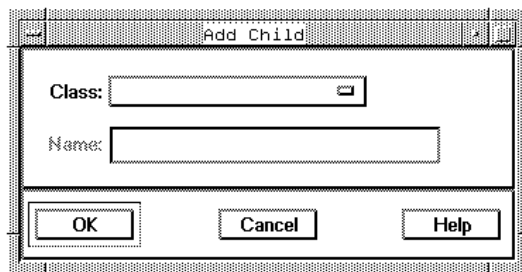
3. Click on **Open** after choosing an object library.

The object library's contents are loaded into the Browser area of the Object Library dialog box.

### Adding an Object to a Library

Add an object to an object library as a child of an existing object. To add an object:

1. Select an object in the Browser area of the Object Library dialog box.
2. Choose **Object** → **Add**. The Add Object dialog box appears.



3. Type the name of the object in the Object entry area.
4. Choose the class to inherit attributes from with the Class option.
5. Type comments about the object, such as creation or purpose, in the Comments entry area.
6. Click on **OK**.

The object is added to the library as a child of the selected object. Choose **Object** → **Edit Source** to edit the new object's methods and events. Choose **Object** → **Properties** to edit the new object's properties.

See Chapter 4, "Using the Browser Tool", for information on cutting, copying, pasting, and adding existing application objects to an object library.

## Deleting an Object

To delete an object from an object library:

1. Select an object in the Browser area of the Object Library dialog box.

2. Choose **Object** → **Delete**. The Delete Object dialog box appears.



Click on Yes to delete the object. Click on No or Cancel to dismiss the dialog box without deleting the object.

Choose **File** → **Revert** to discard edits of an object library. You cannot discard edits after you save the library using the Save, Save As, or Save Turbo options.

When an object is deleted from a library, any children of the object are also deleted.

## Merging Object Libraries

You can consolidate objects in separate libraries by merging them into one library. Copies of objects from other libraries are placed in the current library during a merge. To merge libraries:

1. Choose **File** → **Merge** in the Object Library dialog box. The Open dialog box appears.

Use the dialog box options to search directories and files for the object library. The default search is for files with a .oll file extension. Change the search wildcard if your object libraries have a different file extension.

2. Click on **Open** after choosing an object library.

3. Choose **File** → **Save** to save the added objects in the current library, or choose **File** → **Save As** to create a new object library.

The added objects are visible in the Browser area of the Object Library dialog box when you expand the object hierarchy.

## Saving an Object Library

You can save an object library

- as the current library
- as a new library
- as a turbo version of the library

To save edits in the current library, choose **File** → **Save**. You cannot discard edits after you save the library.

To save the current library as a new library:

1. Choose **File** → **Save As**. The Save As dialog box appears.

Specify a name, location, and permissions for the object library file. You can save the file with any file extension. However, the Builder default search when loading an object library is for files with a .oll file extension.

2. Click on **Save**. The library is saved in the new library file.

An object library saved in turbo form localizes all object classes and resources, creating a deployable library that requires less memory and loads fast. Localizing a loaded object library by default creates a turbo version of the library in an application. See "Using an Object Library" later in this chapter for information on loading and localizing object libraries. To save an object library in turbo form:

1. Choose **File** → **Save Turbo**. The Save As dialog box appears.

---

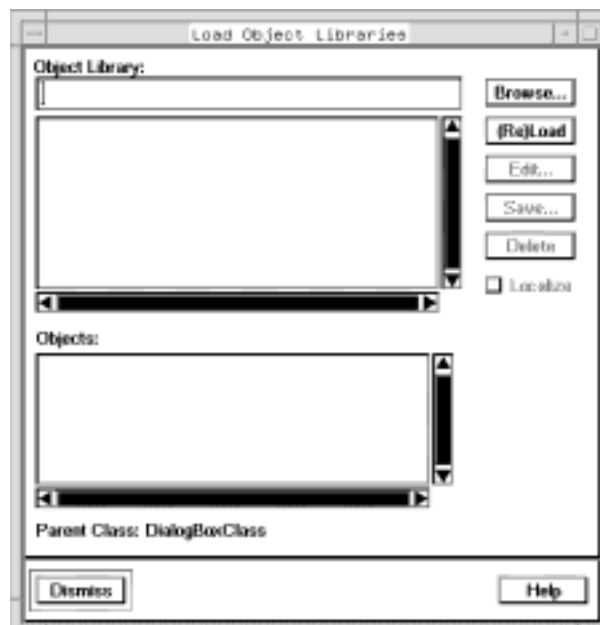
The name of the current library appears in the File entry area with a .olo file extension. Specify a name, location, and permissions for the object library file. You can save the file with any file extension. However, object libraries saved in turbo form by default receive a .olo file extension.

2. Click on **Save**. The library is saved in turbo form.

## Using an Object Library

Use object libraries to decrease duplicate coding and create uniform objects across applications. To load an object library:

1. Choose **Resources** → **Load Objects** from the Browser window. The Load Library Objects dialog box appears.



2. Click on **Browse**. The Load Object Library dialog box appears.



Use the dialog box options to search directories and files for the object library. The default search is for files with a .ol? file extension. Change the search wildcard if your object library files have a different file extension.

3. Choose the object library and click on **Open**. The object library file name appears in the Object Library entry area.
4. Click on **(Re)Load**. The object library file name appears in the Object Library list area, and the objects contained in the library appear in the Objects list area.
5. Turn on the Localize option to save the object library with the application, or turn off the option to keep the object library linked to the file.

6. Repeat steps 2-5 to load additional object libraries. Click on **Dismiss** when you are finished adding object libraries.

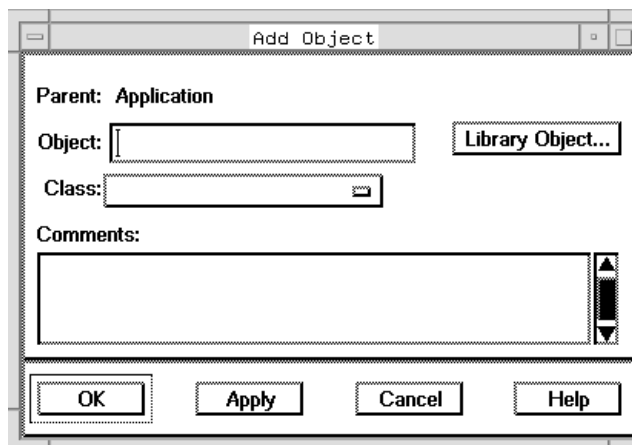
**NOTE:** An application using an external object library does not receive edits from the library until the library is reloaded, or the application is exited and reopened.

Use the Load Object Libraries dialog box to delete object libraries that are no longer needed in the application. To delete an object library:

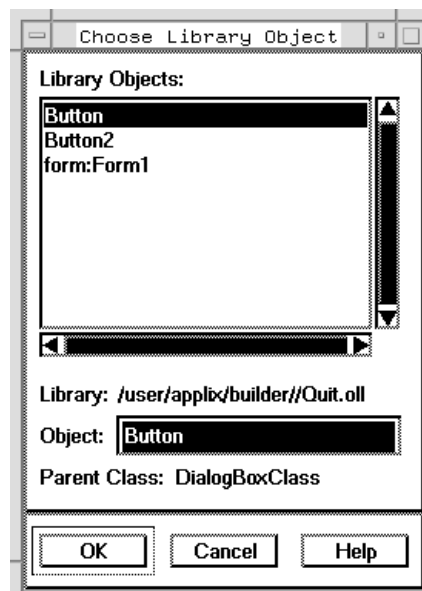
1. Choose **Resources** → **Load Objects** from the Browser window.
2. Click on the object in the Object Library list area.
3. Click on **Delete**.

After an object library is added, the objects in the library can be added to the application. To add an object from an object library to the application:

1. Select an object in the Browser window. The added object becomes a child of the currently selected object.
2. Choose **Object** → **Add** from the Browser window. The Add Object dialog box appears.



3. Click on **Library Object**. The Choose Library Object dialog box appears.



4. Click in the Library Objects list area on the object you want to add to the application.
5. Click on **OK**. The name of the object appears in the Object entry area of the Add Object dialog box.
6. Click on **OK** in the Add Object dialog box.

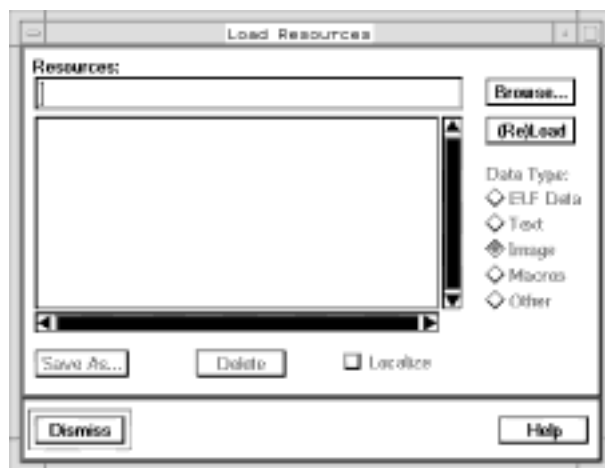
The library object is added to the application. Library objects appear in the Browser window with a rectangle around the object name. You cannot edit the source of a library object. To make changes to a library object you must edit the source of a library object from the Object Library dialog box and reload the object library.

## External Resources

Applications may need resources that are not found in your ELF search path. Use the Resources dialog box to include resources such as an ELF array, bitmaps, ELF macros, or other information in your application. You can use a resource in multiple applications without writing duplicate code or creating copies of the resource for each application.

To add external resources to an application:

1. Choose **Resources** → **Load Resources** from the Browser window. The Load Resources dialog box appears.



2. Click on **Browse**. The Add Resource dialog box appears.

Use the dialog box options to search directories and files for the object library. The default search is for all files. Change the search wildcard if your object library files have a different file extension.

3. Choose the file and click on **Open**. The file name appears in the Resources entry area of the Load Resources dialog box.
4. Click on **(Re)Load**. The file appears in the Resources list area.
5. Choose the resource data type with the Data Type option.
6. Turn on the Localize option to save the resource with the application, or turn off the option to keep the resource linked to the file.
7. Repeat steps 2-6 to load additional resources. Click on Dismiss when you are finished adding external library objects.

Use the Resources dialog box to delete external or localized resources added to the application that are no longer needed in the application. To delete a resource:

1. Choose **Resources** → **Load Resources** from the Browser window.
2. Click on the file in the Resources list area.
3. Click on **Delete**.

Use external resources to add information otherwise unavailable in the application.

## C++ Class Libraries

Applix Builder provides support for using C++ classes in your applications. To use C++ classes in Applix Builder, you need to define the interface to the C++ class in a separate class. The interface is a subclass of the C++ class from which an Applix Builder application can use the methods. This section covers:

- defining the interface class and shared libraries
- loading C++ class libraries
- using C++ classes in an application

### Defining the Interface Class and Shared Libraries

The interface class defines the classes an Applix Builder application can instantiate and the methods in the class that the application can call. You combine the interface class, along with the C++ class files, into a shared library or dynamic link library (DLL) that is supported by your system.

The base interface class and structures for class and member information are defined in the `elfapi.hh` header file, located in the `/install_dir/axdata/elf` directory. This must be defined as the first include file in your interfaces file.

The interface requires you to provide an entry function, `AxGetClassInfo`, that describes the class information table in the shared library. The class information table defines the classes used by the application. This function should return a NULL terminated array of `AxClassInfo_t` structs. This structure contains two fields:

- name of the class
- instance of the class (model object)

For example, the following definition of the `AxGetClassInfo` function defines one class, the `CAXTime` class.

```
AxClassInfo_t *AxGetClassInfo(void)
{
    static AxClassInfo_t class_table[] = {
        {"CAXTime", new CAxtime },
        {NULL, NULL}
    };
    return(class_table);
}
```

The interface class also requires you to define a method information table, `AxQueryInterface`, for each class. The method information table defines the methods used in the interface subclass of the C++ class. This function should return a NULL terminated array of `AxMethodInfo_t` structs. This structure contains three fields:

- name of the method, as it will be used from an Applix Builder object
- a func pointer to the member function
- a NULL terminated ASCII help string that is displayed in the Class Browser as the method's quick help

For example, the following definition of the `AxQueryInterface` function defines the four methods available in the `CAXTime` class.

```
AxMethodInfo_t *CAxTime::AxQueryInterface (void) const
{
    static AxMethodInfo_t method_table[] = {
```

```
        { "setTime", (AX_MEMBER_FUNC)setTime, "Sets the  
initial time" },  
        { "getHour", (AX_MEMBER_FUNC)getHour, "Returns  
the hour" },  
        { "getMinute", (AX_MEMBER_FUNC)getMinute,  
"Returns the minute" },  
        { "getSecond", (AX_MEMBER_FUNC)getSecond,  
"Returns the seconds" },  
        { NULL, NULL, NULL }  
    };  
    return(method_table);  
}
```

The other piece required for the interface is the definition of the interface class, as a subclass inheriting from `CAXInterface` and the C++ class. `CAXInterface` is an abstract class containing virtual functions that need to be defined in the interface class for dynamic binding. The class needs to define the constructors, destructors, method table, and member functions.

The member functions defined in the interface class should correspond to functions that you want to call in the C++ class. All the member functions are of type `elfData`, and take an argument of type `elfData`. You need to explicitly cast the arguments and returned data types of the member functions, using the defined data types in the `elfapi.h` header file, located in the `/install_dir/axdata/elf` directory.

All member functions are passed an argument, even if the argument is `NULL` or is not used by the member function. Likewise, the member function must have a return value, even if the member function calls a `set (void)` method. The member function must return `NULL` if it does not have a return value.

The following example defines an interface class, `CAXDate`, for the C++ class `Date`.

```
class CAXDate : public Date, public CAXInterface {
```

```
public:
    CAxDate() {}          /* Constructor */
    virtual ~CAxDate() {} /* Destructor */
    /* create a similar instance of the class */
    virtual CAxInterface *AxCreateSimilar(void) const {return new
CAxDate; }
    /* method interface table */
    virtual AxMethodInfo_t *AxQueryInterface(void) const;
    /* Member function definitions */
    elfData setDate (elfData args) {
        Date::setDate(AxIntFromArray(args,0), AxIntFromArray(args,
1), AxIntFromArray(args,2));
        return(NULL);
    }
    elfData getYear(elfData args) {
        return(AxMakeIntData(Date::getYear()));
    }
    elfData getMonth(elfData args) {
        return(AxMakeIntData(Date::getMonth()));
    }
    elfData getDay(elfData args) {
        return(AxMakeIntData(Date::getDay()));
    }
};
```

When you are finished defining the interfaces file:

- compile the interface class and the C++ class files
- create a library containing the compiled interface and C++ files

Refer to your compiler documentation for information on compiling and creating shared class libraries or dynamic link libraries (DLL).

## Loading C++ Class Libraries

You can define the C++ class libraries that your Applix Builder application can access, then use the classes in the application. Use C++ object classes to decrease duplicate coding and integrate existing code in your applications. To load a C++ class library:

1. Choose **Resources** → **Load C++ Classes** from the Browser window. The Load Object Classes dialog box appears.



2. Click on **Browse**. The Load C++ Class dialog box appears.

Use the dialog box options to search directories and files for the object library. The default search is for files with a .so file extension. Change the search wildcard if your class library files have a different file extension.

3. Choose the class library and click on **Open**. The object library file name appears in the C++ Class Library Name entry area.
4. Click on **(Re)Load**. The class library file name appears in the C++ Class Library Name list area, and the classes contained in the library appear in the Classes In Selected Library list area.
5. Repeat steps 2-4 to load additional class libraries. Click on **Dismiss** when you are finished adding class libraries.

The classes in the library appear in the Class Browser dialog box, and they are available to define objects in the application.

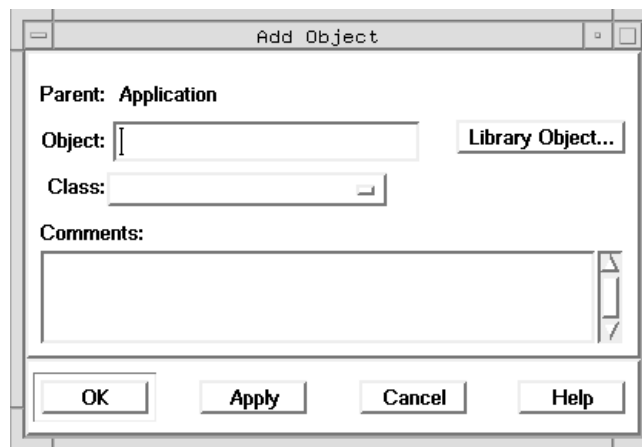
Use the Load Object Classes dialog box to delete class libraries that are no longer needed in the application. To delete a class library:

1. Choose **Resources** → **Load C++ Classes** from the Browser window.
2. Click on the object in the C++ Class Library Name list area.
3. Click on **Delete**.

## Using C++ Classes In an Application

Once C++ classes are loaded into your application, you can create objects based upon the class. To add a C++ object to an application:

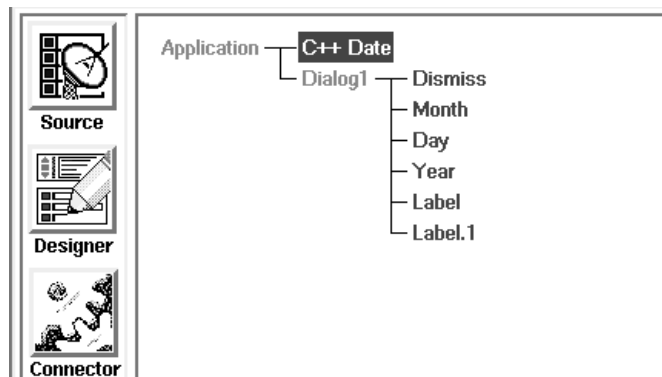
1. Choose the ApplicationClass object in the Browser window.
2. Choose **Object** → **Add**. The Add Object dialog box appears.



3. Type the name of the object in the Object entry area.
4. Choose the C++ class with the Class option.

The class is added to the application as a child of the ApplicationClass object. Use the object just as you would use other Applix Builder or custom objects.

For example, the CAxDate C++ class, loaded as part of a C++ library, can be used to display the current month, day, and year.



The C++ Date object contains the following initialize\_event:

```
#include "datetim_.am"

set initialize_event
    var time, data
    var format date_time_array_ date

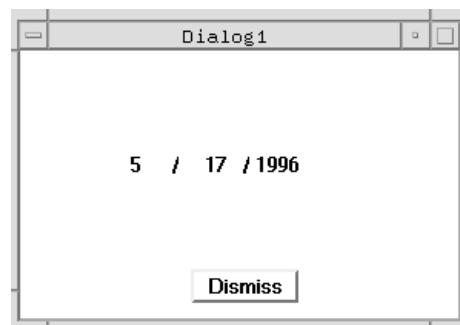
    time    = CURRENT_TIME@()
    date    = DECOMPOSE_TIME@(time)
    data[0] = ("19"++date.year)+0
    data[1] = date.month+1
    data[2] = date.day+1
    this.setDate(data[0],data[1],data[2])
endset
```

The Day, Month, and Year labels in the dialog box have the following initialize\_events:

```
set initialize_event
    var object data
    data = this.application@.child@("C++ Date")
    this.value@ = data.getMonth
endset
set initialize_event
    var object data
    data = this.application@.child@("C++ Date")
    this.value@ = data.getDay
endset
set initialize_event
    var object data
    data = this.application@.child@("C++ Date")
```

```
        this.value@ = data.getYear  
    endset
```

When you run the application the dialog box is displayed with the month, day, and year.



The Date class is defined in the C++ file as follows in the date.h and date.C files:

```
date.h  
#include <iostream.h>  
  
class Date {  
public:  
    Date();          // Constructor  
    void setDate(int, int, int); // year, month, day  
    int getYear();  
    int getMonth();  
    int getDay();  
  
private:  
    int year;  
    int month;
```

```
int day;  
};
```

```
date.C
#include <iostream.h>
#include "date.h"
Date::Date()
{
    year = 1900;
    month = 1;
    day = 1;
}
int Date::getYear()
{
    return(year);
}
int Date::getMonth()
{
    return(month);
}
int Date::getDay()
{
    return(day);
}
void Date::setDate(int h, int m, int s)
{
    if (h<=2050 && h>=1900)
        year = h;
    else
        year = 0;
    if (m<=12 && m>0)
        month = m;
    else
        month = 0;
```

```
    if (s<=31 && s>0)
        day = s;
    else
        day = 0;
}
```

## Distribution and Turbo File Considerations

Loading a C++ class library in an application does not make the classes in the library part of an application. Loading a C++ class only provides access to the C++ class methods. When you create a distribution file or a turbo executable file you must be aware of this connection.

## Remote Objects

Applix Builder allows you to distribute application tasks on different machines in your network using remote objects. Using a remote object you can create a true client/server application.

A remote object can be based on any base or user-defined class. However, a remote object cannot be added as a parent or child of any object in the application. Use a remote object to perform calculations in a SpreadsheetsClass object, create a report in a WordsClass object, or execute any task in your application. A remote object can be a child of another remote object, if both objects are on the same server.

To create and use a remote object you need to:

- Set the DISPLAY environment variable on the server machine to your display
- Set the LD\_LIBRARY\_PATH to the *install\_dir/axdata/axshlib* directory on the server machine
- Start an axnet process on the server machine

The axnet process on the remote (server) machine must be started as a root owned process, independent of Applix Builder, Applixware, or any Applix Builder application.

The axnet process on the host machine must be started using the same path name you use to Applix Builder or Applixware. If axnet is started without the full path name, the macro is unable to find the process and create a remote object.

To start axnet:

1. Have all users on the server exit Applix Builder, Applixware, or any Applix Builder application.
2. Log in as root on the server machine.
3. Add the following line in /etc/services:

**axnet 5492/tcp**

5492 is used to identify the TCP/IP socket for the process.

4. Type ***install\_dir/axdata/axnet &***

where *install\_dir* is the directory where you installed Applix Builder or Applixware and & runs the process as a background process.

The shell window from which you launch axnet logs messages regarding axnet connections and activities. You can leave this window

open to view the messages, which are useful for system administration.

After you start axnet, if you have permission to run on the server machine, with an entry in the `/etc/passwd` file, you can create and use remote objects on the server

Remote object coding uses the following `ApplicationClass` methods:

- `set object_server_close@`
- `set object_server_open@`
- `set remote_object_destroy@`
- `get remote_object_create@`
- `get remote_object_find@`

You can place the code to create a remote object in the `initialize_event` of a local application object, or in any other object event, such as a `ButtonClass` `clicked_event` or a `RealtimeRecordClass` `data_update_event`. Likewise, you can place the code to remove the remote object and close the axnet connection to the server in the `terminate_event` of a local application object, or in any other event. For example, the following lines are sample code in a `ButtonClass` object.

```
@@@ OBJECTS
set clicked_event
  var object obj
  this.application@.object_server_open@("solar")
  obj =
  this.application@.remote_object_create@("solar","WordsClass", "remoteWP")
  /* Execute WordsClass Tasks */
  obj.is_windowless@ = FALSE 'display remote window
  obj.type@("Type this in a remote WordsClass object")
endset
```

```
set terminate_event
  var object obj
  obj =
this.application@.remote_object_find@("solar","remoteWP")
  if obj <> NULL
    this.application@.remote_object_destroy@("solar",obj)
    this.application@.object_server_close@("solar")
endset
```

In the example, solar is the name of the server, WordsClass is the class the remote object inherits attributes from, and remoteWP is the name of the remote object.

## Common Dialog Boxes

Applix Builder provides common dialog boxes that you can use to present a consistent user interface throughout your applications. The common dialogs are accessed as methods in the CommonDlgClass. The common dialog box methods are:

set run_application_dlg@	Displays Applix Builder Run dialog box
set run_macro_dlg@	Displays Run Macro dialog box
set send_mail_dlg@	Displays Send Mail dialog box
get choose_bitmap_dlg@	Displays Icon Viewer dialog box
get choose_color_dlg@	Displays Choose Color dialog box

get open_dlg@	Displays Open dialog box
get print_dlg@	Displays Print dialog box
get save_dlg@	Displays Save dialog box

The ApplicationClass is an inherited class of the CommonDlgClass, all the common dialog box methods are also available through the ApplicationClass. For example, the following is a sample ButtonClass clicked\_event that displays the Open dialog box.

```
set clicked_event
    var object app
    var file
    app = this.application@
    file = app.open_dlg@
/* Use the returned file name in the application */
    ...
endset
```

The open\_dlg@ method returns the full path name of the selected file. You can use the returned file name to open a file.

Use the Class Browser dialog box to get help on the CommonDlgClass methods.


## Applix Builder Bitmaps

When you run Applix Builder the bitmaps used by the various components are cached in the system. You can use the bitmaps to

customize your ExpressLine, applications, or even your Applixware main menu. The following are the IDs for several Builder bitmaps.

23-18x23  ExpressLine icon

23-50x50  50x50 icon

ar50x50  50x50 color icon

Applix Builder bitmap IDs begin with 23. Applix Builder color bitmap IDs begin with ar\_. Use the CommonDlgClass get method `bitmap_dlg@` to display the Icon Viewer dialog box. Use the dialog box to get the names of all available bitmaps cached in the system.

---

# 9 CORBA Support

---

Applix Builder provides support for CORBA objects. CORBA is a standard for distributed object technology that allows you to use objects throughout your network, even if they have been written in different languages, or reside on workstations running different operating systems.

This chapter covers the following topics:

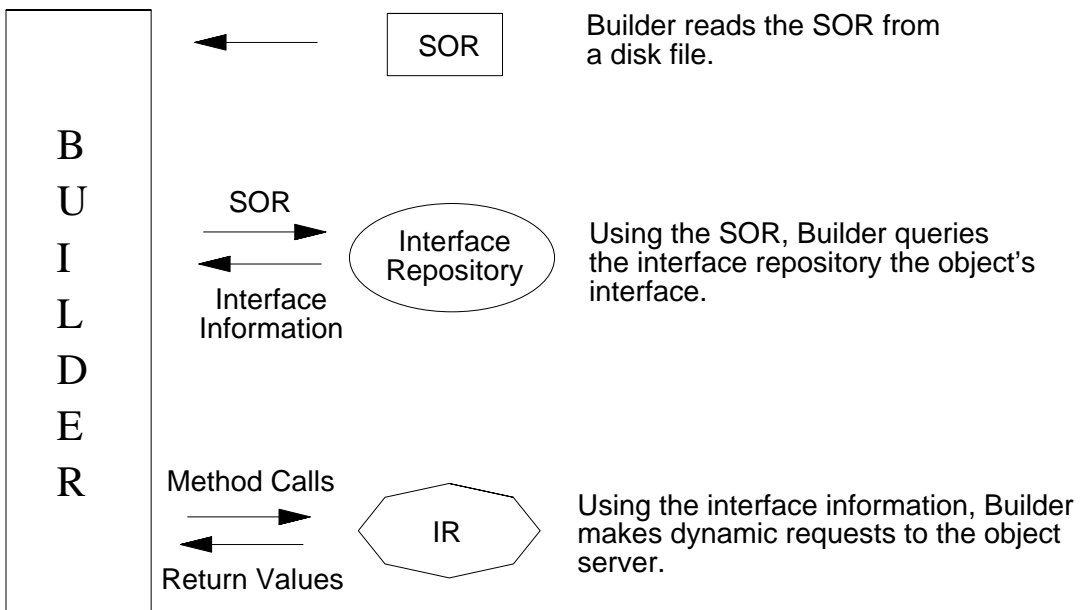
- CORBA Architecture
- Using CORBA Objects
- Using Builder Objects as CORBA Objects

## CORBA Architecture

Common Object Request Broker Architecture (CORBA) is an industry standard that allows an application to use objects developed in a variety of programming languages, and residing on separate machines in a network.

For the Builder Programmer, CORBA allows you to access useful objects residing in any part of your network. These objects may exist on the same machine as your Builder application, or they may exist on a different machine. The objects may be implemented in C++, Smalltalk, or some other programming language supported by your ORB's IDL (Interface Definition Language) compiler.

At the center of any CORBA distributed application is an ORB (Object Request Broker). Applixware supports any CORBA 2-compliant ORB that supports the IIOP protocol. The following figure shows the architecture of a Builder application accessing a CORBA object.



## CORBA Terminology

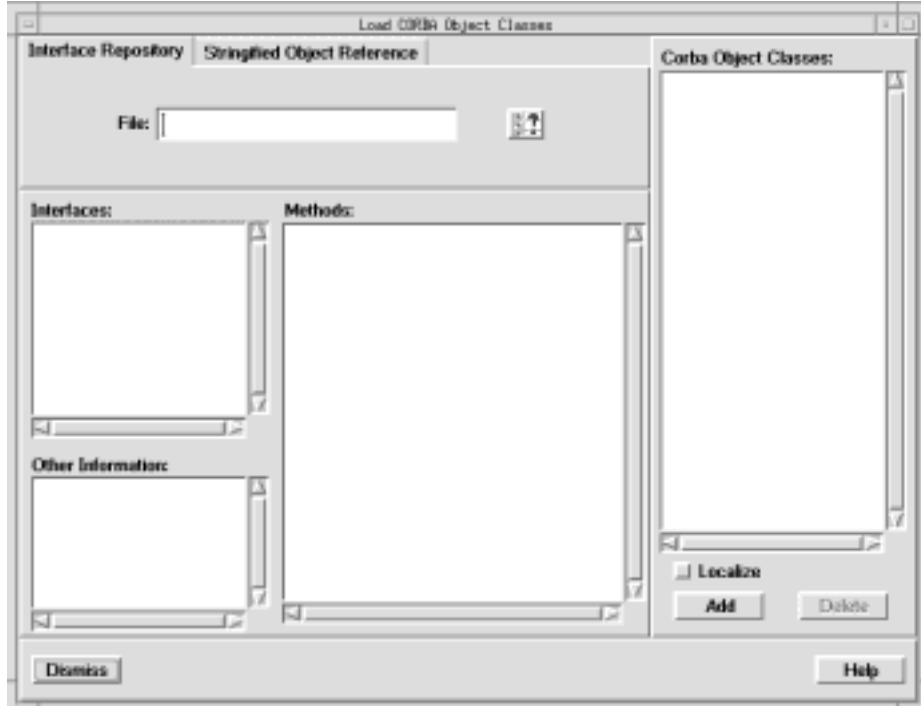
This chapter uses a number of terms that are specific to CORBA development environments:

- An *ORB* (Object Request Broker) acts as an arbiter between client applications and object servers.
- An *Interface Repository* is a database with which objects can register. Once an object registers with the database, any client can query the Interface Repository to obtain information about the object's interface.
- A *Stringified Object Reference* (SOR) is a unique identifier of the CORBA object server in string form.
- An *Object Server* implements an object's interface.

## Using CORBA Objects

Builder allows you to retrieve a list of CORBA objects from your ORB's Interface Repository. To do this, follow these steps:

1. Run Builder.
2. From the Builder main menu, choose **Resources** → **Load CORBA Classes**. The Load CORBA Object Classes dialog box appears.



There are two tabs in this dialog box:

- The **Browse IR** tab allows you to retrieve information on all objects registered with the Interface Repository.
- The **Use IOR** tab allows you to query and create a Builder object from a Stringified Object Reference.

3. Click **Browse IR**.
4. Enter the name of a file containing the IOR of the interface repository in the file name field.
5. Click the Query button.



The name of the objects that have been registered with the interface repository are displayed in the Interfaces list box.

6. Select an object from the Interfaces list box. Information about the object appears in Other Information and Methods list boxes.
7. Click **Add** to create a Builder class corresponding to the interface selected.

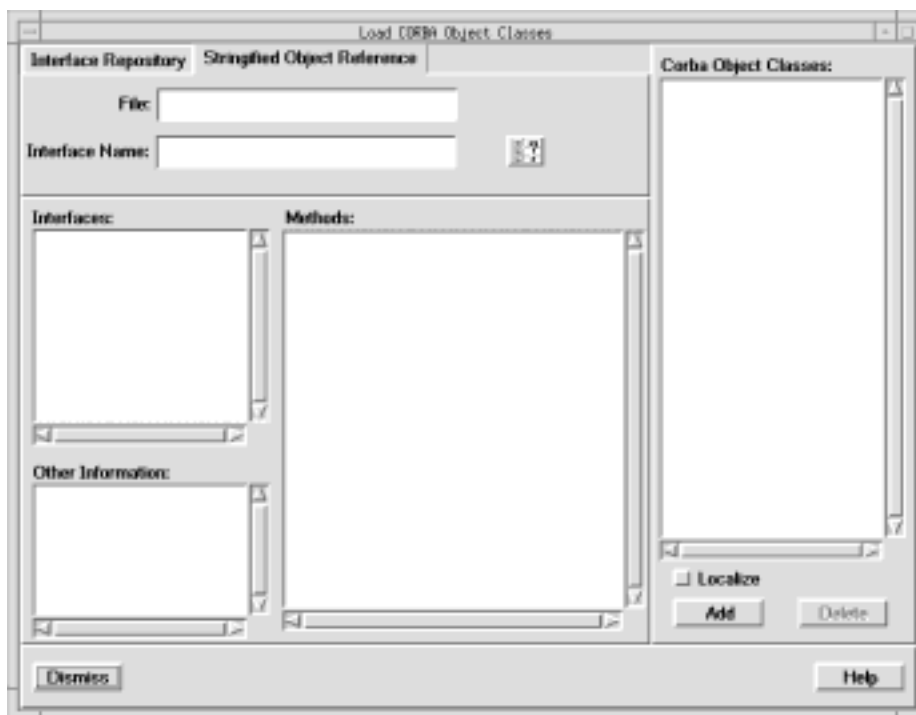
**NOTE:** No Builder object is created. To get a CORBA object that implements the newly-created interface, go to the Use SOR tab, and follow the instructions in the next section.

## Using a CORBA Object Reference

If your server administrator gives you a stringified object reference to a CORBA object, you can use it to generate an object in Builder. This object corresponds to a single CORBA object. You cannot use a stringified object reference to build two CORBA objects in the same Builder application. The stringified object reference is typically written to a disk file, then read from the disk during Builder development.

To add a Builder object using a stringified object reference, follow these steps:

1. Run Builder.
2. Open the application to which you want to add the new object.
3. Choose **Resources** → **Load CORBA Classes**.
4. Click the **Stringified Object Reference** tab. The following dialog box appears:

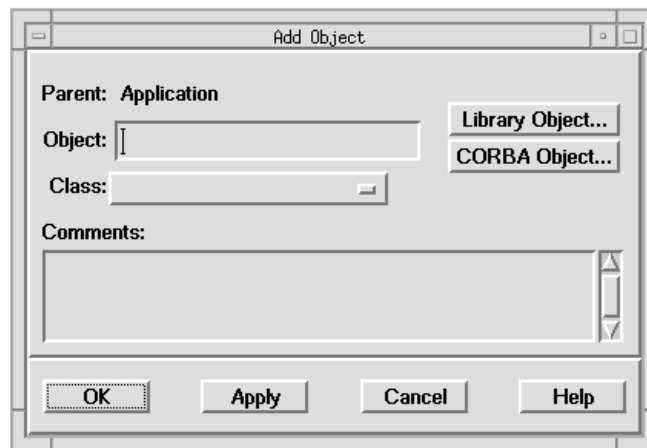


A stringified object reference is typically stored in a disk file on your local file system.

5. Enter the absolute pathname of the file containing the stringified object reference in the File field.
6. Click **Query**. The interfaces, methods, and data structures for the object appear in the dialog box.

**NOTE:** If the information is not returned from the interface repository, enter a name in the Interface Name field. This name will be used by your builder application.

7. Click **Add**. This creates a Builder class corresponding to the interface, and a Builder object corresponding to the CORBA object with the given SOR.
8. Click **Dismiss**.
9. From the Builder Application Window, choose **Object** → **Add**. The Add Object dialog box appears.



10. Click **CORBA Object**. The Choose Library Object dialog box appears. When this dialog box first appears, it has the entire CORBA object reference file name in the CORBA Object field.
  11. Change the entry in the Object field to a name and click **OK**.
  12. Press **OK** on the Add Object dialog box.
- The object is added to your application.

## Builder Programming Issues

Programming in Builder with CORBA objects is very similar to programming with native Builder objects. This section describes some similarities and differences when programming with CORBA objects.

### Calling CORBA Methods

You call CORBA methods in the same way that you call Builder methods. No data conversion either from Builder to C or from C to Builder is necessary.

**NOTE:** If the CORBA method is an ELF keyword, you can call the method by using the ! notation. For example:

```
' the CORBA Object method name is "set", which is a keyword in Elf.
var methodStr
methodStr = "set"
this.!methodStr(1,2,3) ' actually calls the
                        ' "set" method on the CORBA object
```

### Invocation Mode

All CORBA object classes inherit from the CORBA class. The CORBA class, has a method called `invocation_mode@`, which allows you to call CORBA methods in one of three ways:

- Synchronous - If you set `invocation_mode@ = PENDMODE#INVOCATION`, the Builder application calls the method, then waits for the method call to be resolved. The application is blocked while the request is being processed. This is the default.

- One Way - If you set `invocation_mode@ = ONEWAY#INVOCATION`, the Builder application calls the method, then continues its own processing. This invocation method is used when the application is not interested in the results of the method call.
- Deferred Synchronous - If you set `invocation_mode@ = DEFERRED#INVOCATION`, the Builder application calls the method, and receives a request ID. The application can then continue processing. At some point in the future, it calls the method `deferred_request_response@` passing the request ID as an argument. The data returned by `deferred_request_response@` is the result of the method call.

Once you set an invocation mode for an object, the mode stays the same until it is changed. You can set different invocation modes for different objects in an application.

**NOTE:** The invocation mode constants `PENDMODE#INVOCATION`, `ONEWAY#INVOCATION`, and `DEFERRED#INVOCATION` are defined in the ELF include file `builder_.am`.

## Class Inheritance

When a CORBA object class is created in Builder, it has in it all the methods supported by the object, and the interfaces from which it inherits.

If you add a CORBA class that inherits from a parent class, all the methods supported by the parent class will be present in the child class.

## Formats and Structs

When you edit a Builder object created from a CORBA object, the method window automatically includes any ELF formats and arrays that correspond to the structures defined in the CORBA object's interface.

## Data Mapping

Data in a CORBA IDL file is returned to Builder as various types of ELF data. The mapping between CORBA data types and ELF data types is shown in the following table.

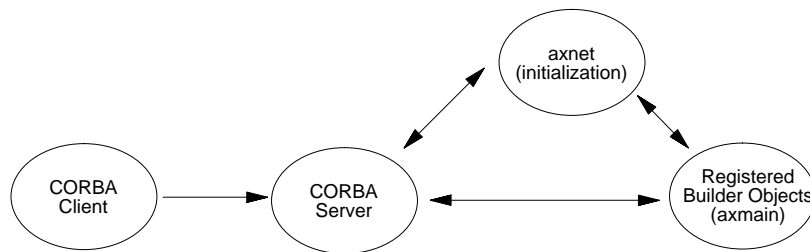
<b>CORBA Type</b>	<b>Builder Data Type</b>
boolean	ELF TRUE (-1) ELF FALSE (0)
char	string
octet	ELF binary
string	string
short	numeric
long	numeric
float, double	numeric
enum	numeric
struct	format
union	format array[0] is the discriminant array[1] is the value
sequence	array
array	array
any	format array[0] is the structure name or data type array[1] is the value
object	Builder Object

Exceptions are returned as ELF errors.

Out and inout parameters in a CORBA IDL file are returned to Builder as ELF arrays.

## Using Builder Objects as Corba Objects

You can create a Builder object that acts as a CORBA server object. Builder objects that act as CORBA server objects can be called from a CORBA client program, and provide all the functionality built into the Builder application to that CORBA client. The following figure shows the architecture of the Applixware/CORBA environment:



In this figure, axnet and axmain must be running on the same physical machine. The CORBA client and CORBA server can be running on different machines.

The CORBA server is built from parts generated by Builder, and the ORB IDL compiler. The axnet process is used to initiate a session between the CORBA server object and axmain. Once the session is

started, communication goes directly between the CORBA server object and the registered Builder objects.

## The Applixware CORBA Server Environment

Before you can use a Builder application as a CORBA server, you must be sure the following conditions exist:

- Axnet must be running on your workstation. The best way to start axnet is to run Applixware on your workstation, but you can also run axnet separately. To do this, log in as root, change directory to `/install_dir/axdata`, and enter this command:

```
./axnet &
```

See the *Applixware Linux System Administrator's Guide* for more information on axnet.

- The objects in the builder application must be registered. To do this, each method must call the macro `REGISTER_OBJECT@` in its `initialize_event`. For more information on this macro, see the following section.
- You must initialize the Applixware CORBA server by running the macro `INITIALIZE_CORBA_SERVER@`. For more information on this macro, see the following section.
- The CORBA server code must be linked with `elfapi.a`. It must call the function `AxInitializeBuilderObject` to initialize the object, `AxCallBuilderObjectMethod` to use the object, and `AxTerminateBuilderObject` to end the session. See the section "Coding the CORBA Server", later in this chapter, for more information.

## Applix Object Registry

The Applix object registry is contained in the file `bldrObject.reg`. This file is written to your `/axhome/macros` directory in your home directory.

You can change the location of this file by setting the Applix preference `bldrRegisterObjectsFile`. By setting this preference to the same path for a set of users, you can have those users share the same registry information.

## Applixware Object Registration Macros

Applixware supports a set of macros that allow registration of persistent objects. CORBA client programs can only access Builder objects that have been registered with Applixware. This section describes those macros.

### INITIALIZE\_CORBA\_SERVER@

<b>Format</b>	INITIALIZE_CORBA_SERVER@()
<b>Description</b>	This macro registers your Applixware with axnet, and initializes the Applixware object registry. The axnet process must be running on your workstation.

## REGISTER\_OBJECT@

**Format** REGISTER\_OBJECT@(object, interface\_name, overwrite\_flag, path\_to\_ab\_file)

**Arguments** object - An object reference to the object you want to register.

interface\_name - A string name. This name is used by other applications to retrieve the registered object from the Applixware object registry.

overwrite\_flag - A Boolean. If TRUE, this function will overwrite any object in the Applixware registry with the same interface name. If FALSE, an entry in the Applixware registry with the same interface name will not be overwritten.

path\_to\_ab\_file - A string containing the absolute path name of the Builder file containing the object you are trying to register.

**Description** This macro registers an object with the Applixware object registry. Typically, you register an object by calling a Builder method in the set initialize\_event of the object you want to register. For example, here is an example initialize event for a list box object:

```
set initialize_event
,
,       Register the object using the name ListMe
,
register_object@(this,"ListMe", TRUE,
"/user/applix/builder/list_box.ab")

endset
```

## UNREGISTER\_OBJECT@

<b>Format</b>	UNREGISTER_OBJECT@(object, interface_name, delete_entry)
<b>Arguments</b>	object - An object reference to the object you want to unregister.  interface_name - A string name. The name of the object in the object registry.
<b>Note</b>	You can provide either the name or the object, and pass the second argument as a NULL.
<b>Description</b>	This macro removes the named object from the Applixware object registry.

## FIND\_REGISTERED\_OBJECT@

<b>Format</b>	FIND_REGISTERED_OBJECT@(object_name, startFlag)
<b>Arguments</b>	objectName - An object reference to the object you want to find, or a string containing the name of a registered object.  StartFlag - A Boolean. If TRUE, the program containing the object is started when it is found.
<b>Description</b>	Searches the Applixware object registry for an entry matching the objectName argument. If it finds the object, it returns an object reference. If Start_Flag is set to TRUE, Applixware starts the program containing the named object.

## INITIALIZE\_OBJECT\_REGISTRY@

<b>Format</b>	INITIALIZE_OBJECT_REGISTRY@()
<b>Description</b>	Initializes the Applixware object registry. This function does not register Applixware with axnet. This function should only be called when you want to make registered objects available, but are not using them as CORBA objects.

## Callbacks

Builder supports callbacks from CORBA server objects. When you register a Builder object as a callback, the Builder application listens on a special port for a message from the CORBA server. When the CORBA server send a message, the registered callback object is called.

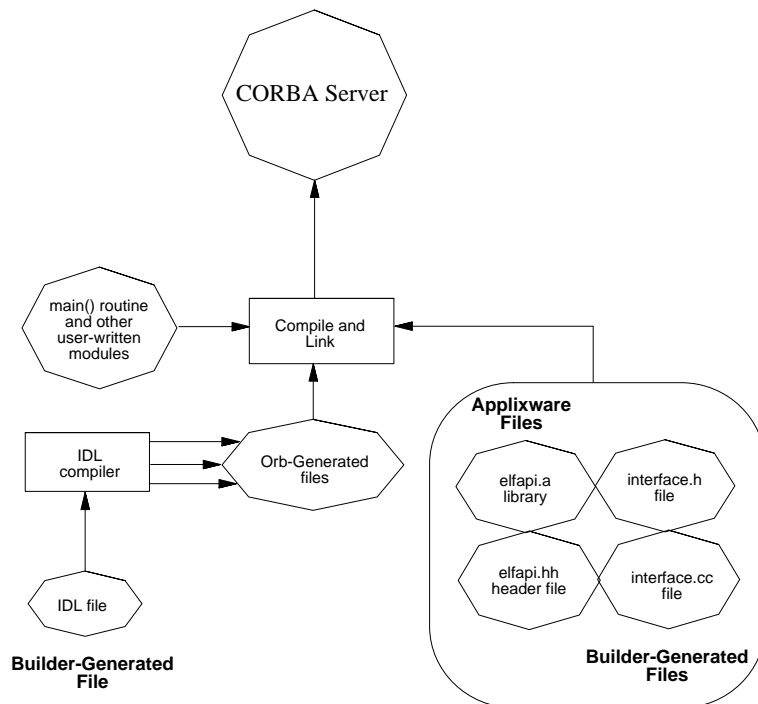
To register a builder object as a callback, you must call the following macro:

## REGISTER\_CALL\_BACK\_OBJECT@

<b>Format</b>	register_call_back_object@(builderObject, classLibraryObject)
<b>Arguments</b>	<p>builderObject - An object reference to the object that will receive the callback.</p> <p>classLibraryObject - An object reference to a CORBA Class Library object. The interface for this class is used when unpacking data sent from the CORBA server.</p>
<b>Description</b>	Registers a Builder object as a callback. The Builder application listens on a port for a response. When a message arrives, the method called by the CORBA server for builderObject is called.

## Creating CORBA Objects in Builder

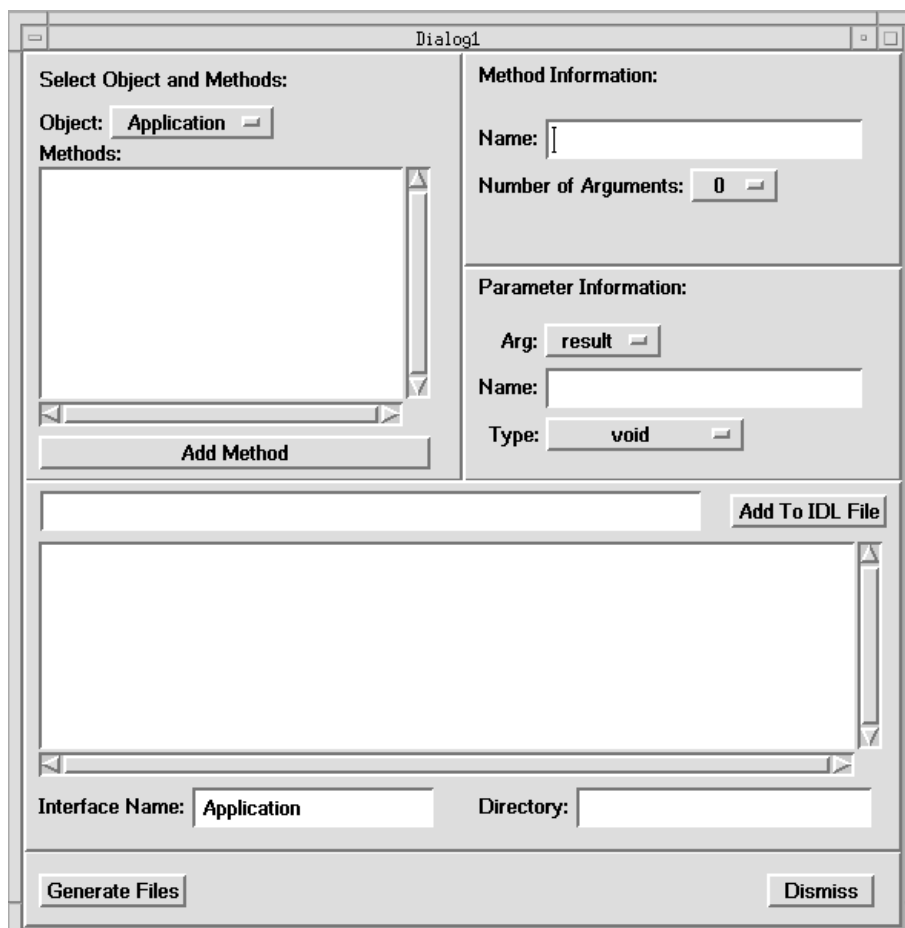
To export Builder objects as CORBA objects, you must have a working Builder program. You can write a new one, or use any existing Builder program. The following figure shows how a CORBA server is built.



Follow these steps to export parts of your Builder application as CORBA objects:

1. Open the Builder application.

- From the Builder main menu, choose **Tools** → **Create CORBA Objects**. The Create CORBA Objects dialog appears.



The names of the objects in the Builder application appear in the Object option button.

3. Select the name of the object you wish to use as a CORBA object from the Object option button. The object's methods appear in the list box.
4. Click the name of a method in the Methods list box.
5. Click **Add Method**. The name of the method appears in the Name field.

A prototype showing the CORBA method call and the arguments appears in the field below the Add Method button. By default, the call returns a short, and accepts two arguments, both shorts. For example:

```
short mortgage(in short arg1, in short arg2);
```

6. Use the # of arguments option buttons to select the number and type of the arguments accepted by the method call. You can also establish the type of result the method returns. For example, you can set the result to return a string or a void.

**NOTE:** The rest of this procedure shows how to set up a method to take three arguments: a float, a long, and a string.

7. Click on the # of arguments method, and change that value to 3. The method call changes to display three arguments:  

```
short mortgage(in short arg1, in short arg2, in short arg3);
```
8. Under Parameter Information, click the **Arg** option button, and change that option from result to arg1.
9. Click the **Type** option, and change that option to a float.
10. Click the **Arg** option button, and change that option from arg1 to arg2.
11. Click the **Type** option, and change that option to a long.

12. Click the **Arg** option button, and change that option from arg2 to arg3.
13. Click on the **Type** option, and change that option to a string.  
short mortgage(in float arg1, in long arg2, in string arg3);
14. Click **Add to IDL File**. The constructed method appears in the list box at the bottom of the screen.

## Generating Files

Once all the methods that you want to expose to CORBA applications appear in the list box at the bottom of the screen, follow these steps to generate the CORBA stub files:

1. In the Interface Name field, enter the name that you wish to assign to the files. For example, if you enter mortgage, the files generated all contain the string mortgage.
2. Enter a directory name in the Directory field. This directory must exist in your file system.
3. Click **Generate Files**. Three files are generated:
  - The implementation file is named *<Interface\_name>\_i.cc*. For example, the implementation file would be called mortgage\_i.cc.
  - The header file is named *<Interface\_name>\_i.hh*. In this example, the header file would be called mortgage\_i.hh.
  - The IDL file is named *<Interface\_name>.idl*. In this example, the IDL file would be called mortgage.idl.

User-defined type, such as structs and arrays, are not generated automatically. You must provide the code to convert from ELF for these arguments or return values.

An Interface Definition Language file is written in the Interface Definition Language IDL. The following example shows the Interface Definition of a Mortgage object:

```
// Mortgage.idl
// IDL definition for Mortgage

interface Mortgage {
    short mortgage(in float arg1, in long arg2, in string arg3);
};
```

## Coding the CORBA Server

Using your ORB's IDL compiler, generate the stub files and write the main() routine for the server. Compile this main() routine, along with the implementation files. Include the file elfapi.hh, and link with elfapi.a to generate your CORBA server. The elfapi.a library and the elfapi.hh header file is in /install\_dir/axexec/axdata/elf.

Elfapi.a contains functions that are required for a CORBA client program to call a registered Applixware object:

- int AxInitializeBuilderObject (char \*name, char \*axmain\_host)

The name argument should match the registered name of the Applixware object you are trying to call.

The axmain\_host argument should be the name of the host machine on which the CORBA server is running.

This function returns an id. This id is required in calls to other functions.

- void AxTerminateBuilderObject (int id)

The id is the one returned from the function `AxInitializeBuilderObject`.

This function terminates the session with the registered object.

- `elfData AxCallBuilderObjectMethod AXP_((int id, char *methodName, elfData args, int *error, char **errStr));`

The id argument is the one returned by `AxInitializeBuilderObject`.

The methodName argument is the name of the method to call.

The argarray is an elf data array containing the arguments to pass to the method.

The error argument is a pointer to an integer. If the method returns an error, this pointer contains the error number.

The errString argument is a pointer to a pointer to a string. This handle is used to return an error string, should the method call return an error.

See your ORB documentation for more information on coding requirements for your CORBA server.

---

# 10 Building User Interfaces

---

You can create custom applications with Applix Builder. These applications will run on Applix Anyware.

In order to deploy these applications across diverse platforms, you must be care to design the user interface portions of these applications according to specific guidelines. This chapter describes those guidelines.

This chapter covers the following topics:

- Designing dialog boxes
- Implementing applications
- Customizing the Anyware Main Menu

## Designing Dialog Boxes

The dialog boxes you design for Anyware applications run on many different browsers and operating systems. This section covers basic design principles for dialog boxes. Use these design principles to provide consistent application appearance across the different browsers and operating systems. This section includes:

- Dialog box guidelines
- Form guidelines
- Using fonts

Dimensions in pixels refer to application dialog boxes created in Applix Builder.

### Dialog Box Guidelines

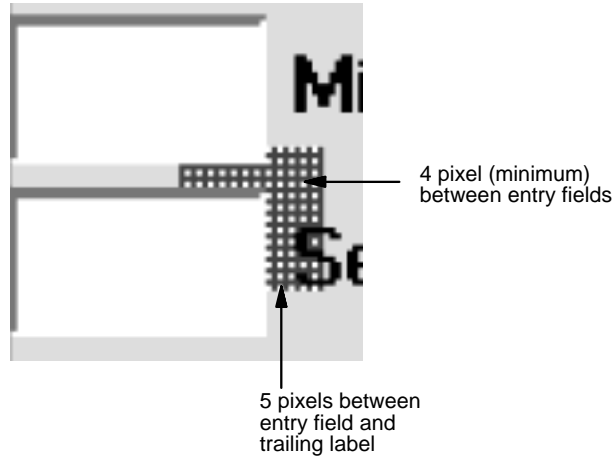
These guidelines address the spacing requirements of Applix Builder dialog boxes and controls in dialog boxes.

- The dialog box should be no larger than 775 pixels wide by 560 pixels high. The dimensions, plus the window manager decoration and Java applet banner appended to the dialog box, are the maximum allowed for displaying a dialog box on an 800 pixel wide by 600 pixel high screen.

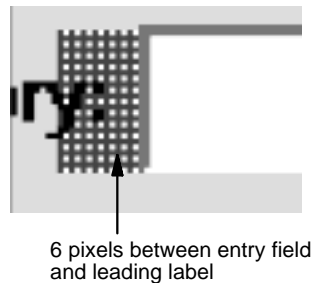
To prevent control overlap in a dialog box, use the following guidelines:

- Maintain a space of at least 10 pixels between controls and the edge of a dialog box or panel.

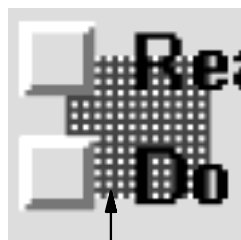
- Maintain a space of at least 4 pixels between entry fields. When an entry field is selected in Anyware a 2 pixel outline box highlights the selected entry field.



- Maintain a space of at least 5 pixels between an entry field and a separate trailing label. Maintain a space of at least 6 pixels between an entry field and a separate leading label. If the title is displayed as part of the entry field the spacing is set automatically.

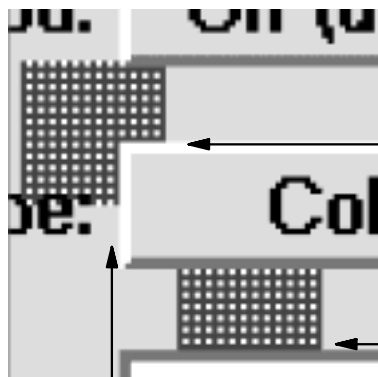


- Maintain a space of at least 7 pixels between a toggle button and a separate trailing label.



7 pixels between toggle button and label

- Maintain a space of at least 13 pixels between option buttons or combo boxes.



13 pixels between option buttons (combo box)

14 pixels between option buttons (combo box) and entry fields

5 pixels between label and option button (combo box)

- Maintain a space of at least 14 pixels between an option button or combo box and an entry field.
- Maintain a space of at least 5 pixels between an option button or combo box and a separate leading label.

## **Form Guidelines**

To prevent control overlap in a form, use the following guidelines:

- All labels should be right justified.
- Fields or labels should be at least 120 twips from the top of the form.
- Maintain a space of at least 60 twips between entry fields.
- Maintain a space of at least 60 twips between an entry field and a separate leading label.
- Maintain a space of at least 120 twips between a multiline box and an entry field below the box.
- Query arrow buttons should be 300 by 300 twips.
- Text boxes and check boxes should have a height of 300 twips.
- A combo box should have a height of 315 twips.
- You can have only a vertical option box, not a horizontal option box.

## **Implementing Applications**

You can program an application in Applix Builder that can run as either a standard Applixware application or as an Anyware application. The minor differences between programming for Applixware and Anyware are covered in this section. When implementing an application for Anyware you need to consider:

- ELF macros for use with Anyware

- Menu bar programming

## Anyware ELF Macros

Most of the ELF macros and Applix Builder methods can be used in Anyware. There are additional macros available to use specifically with Anyware applications. These macros are:

- ANYWARE\_HEARTBEAT\_RATE@
- ANYWARE\_REMOTE\_INTERFACE@
- ELF\_STACK@
- JAVA\_COMMS\_WINDOW@
- PRINT\_DATUM@

The macros are of most use when you are either debugging a program, or initializing runtime features. See "The javalogin.am file" in Chapter 1 for information about setting runtime features.

### ANYWARE\_HEARTBEAT\_RATE@

The Anyware Server sends a message to an idle client after a preset time in order to maintain a server connection. By default the time is set to 45 seconds, but you can change the time interval with the ANYWARE\_HEARTBEAT\_RATE@ macro. You may want to change the time interval of the message if your web server timeout is shorter than the Anyware default value.

The macro takes the time interval, in seconds, as an argument. Use the macro in your javalogin.am macro to change the time interval so that it does not exceed the server time-out for closing the connection.

## ANYWARE\_REMOTE\_INTERFACE@

You may deploy applications that are used both in Applixware and Anyware. Use the ANYWARE\_REMOTE\_INTERFACE@ macro to determine in which environment your application is running. You can design your application to change or disable application features, or perform different login routines depending on the environment.

The macro returns a Boolean value. The macro returns TRUE if the application is running in Anyware as a Java client. The macro returns FALSE if the application is running in Applixware, appearing on a local display.

## ELF\_STACK@ and PRINT\_DATUM@

An Anyware application appears on a client display, but the processing actually occurs windowlessly on the Anyware host. Debugging an Anyware application requires presenting the information on the display of the server machine running the Anyware axmain processes. You can debug an application in the Macro Editor debugger on the server display by calling the following macros, in order, from your Anyware application.

- ME\_APPLICATION\_DLG@
- ME\_TOGGLE\_DEBUGGER@

The first macro launches a Macro Editor window, then the second macro places the Macro Editor window in debugger mode.

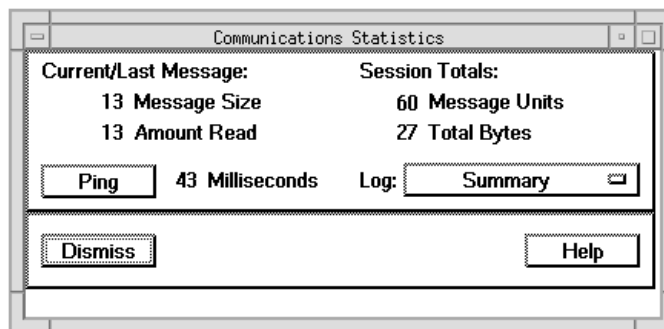
The ELF\_STACK@ and PRINT\_DATUM@ macros also provide application information, in addition to launching a debugging session.

The PRINT\_DATUM@ macro prints the contents of the passed ELF datum to the standard output on the server where the client's axmain process is running.

The ELF\_STACK@ macro returns an array of strings containing the ELF execution stack. Use the macro to determine how the current function was called.

## JAVA\_COMMS\_WINDOW@

Anyware has a Statistics dialog box you can use to monitor communication between your application and the Anyware Server. Choose Tools → Display Status Window in the Anyware Main Menu.



Use the JAVA\_COMMS\_WINDOW@ macro to display the dialog box from your application. You may want to use the dialog box with your application to enable session logging, or to determine the amount and frequency of communication between your application and the server.

## Menu Bar Programming

When you set the state of menu bar features in a Builder application deployed in Anyware, you cannot use the MenuBarClass

`menu_state_event` and `menu_name_event`. There are two set methods you can use in your application to set the status and display string of menu bar items:

- `menu_item_name@`
- `menu_item_status@`

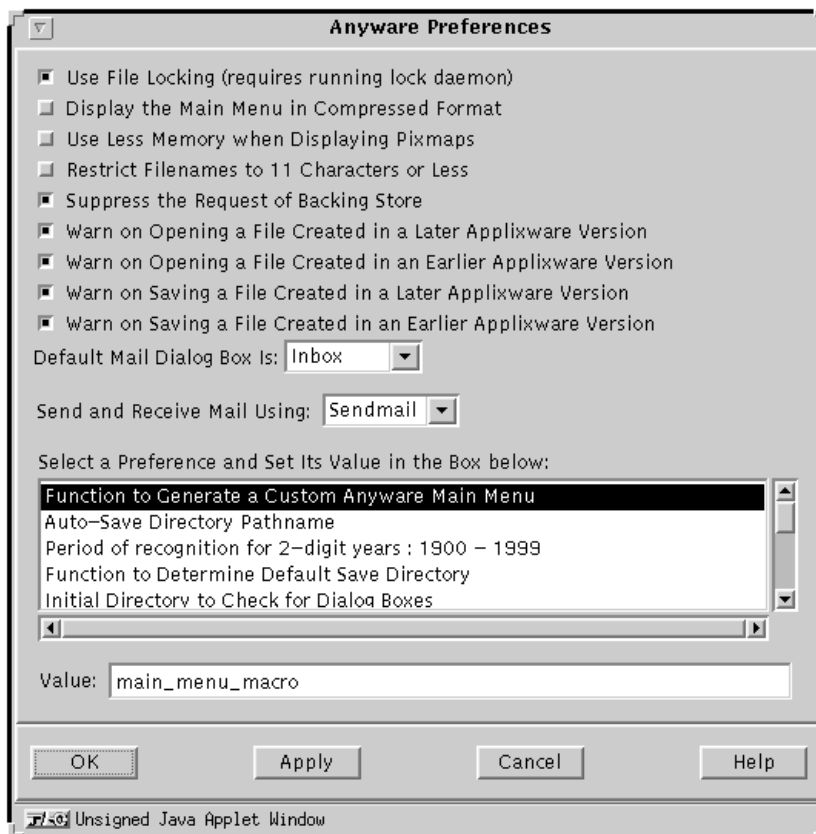
The `menu_item_name@` method takes two arguments: the function name and the string to display in the menu bar for the function. Use the method to create dynamic text strings for a menu choice, such as an undo feature.

The `menu_item_status@` method takes two arguments: the function name and the display status for the function in the menu bar. Use the method to set the status of a menu bar choice, making it available, grayed, or even a toggle or radio button. The menu bar status values are defined in the `/install_dir/axdata/elf/menubar_.am` header file. Include the header file in the method source where you make calls to this method. The status values are defined as follows:

<code>MENUSTAT#NORMAL</code>	Display normally
<code>MENUSTAT#DIMMED</code>	Display grayed
<code>MENUSTAT#TOGGLE_ON</code>	Toggle on to left
<code>MENUSTAT#TOGGLE_OFF</code>	Is a toggle, but off now
<code>MENUSTAT#RADIO_OFF</code>	Is a radio, currently off
<code>MENUSTAT#RADIO_ON</code>	Radio on to left
<code>MENUSTAT#NO_SHOW</code>	Suppress display altogether

You cannot hide menu bar items in an Anyware application. To disable menu bar items, you must gray them. Setting the status of a menu bar choice to `MENUSTAT#NO_SHOW` grays the choice.

Menu items that are displayed as toggle or radio buttons need to be initialized before they are displayed in the application.



1. Exit the Applixware Iconbar, then redisplay it.

You do not have to exit entirely out of Anyware to test the Anyware Iconbar macro.

## **Customizing the Iconbar**

Because the Anyware Iconbar is implemented as a Builder application, the menu bar attempts to execute the functions as methods, unless otherwise noted. To add a macro function to a menu bar, the macro name should be preceded by an @. For example, to call the macro my\_function as an item, the name should appear as

@my\_function

in the Method entry area of the Create Menu Bar dialog box. A customized Anyware Iconbar is saved as aa\_mm4 in the user's axhome directory.



---

# Glossary

---

**application** An operational program or set of programs intended to perform a set of tasks.

**Applix Builder** A graphical application development environment with object-oriented tools and features.

**Applixware** A suite of comprehensive applications designed to help you with many of your daily work tasks, such as writing letters and reports, mailing documents, sending faxes, creating illustrations and technical drawings, and performing data analysis.

**attributes** The inherent and distinctive qualities that define an object or class. An attribute can be a size, shape, or color, as well as an abstract or hidden component that is not apparent to the application user.

**bitmap** A graphic image used as a dialog box decorative element.

**broadcast** An invocation of a method in one or more objects using \* notation. A broadcast is a simpler and more efficient way of invoking methods in an array of objects, instead of writing a loop to reference each array item. A broadcast also allows invoking a method for a set of objects that cannot be de-referenced as an array.

**call inherited** A direct call of an inherited method, used when an object has a method that overrides an inherited method.

**canvas** A drawable area in the dialog box. Use a *pen* to draw text and graphical objects upon the canvas.

---

**child** An object that belongs to another object. In Applix Builder a child has unique properties apart from the *parent*, the object it belongs to. The relationship of children and parent objects is based upon enclosure, not *inheritance*.

**class** An abstract definition of the operations used to implement functionality. A class describes the expected behavior and attributes of a specific instantiation of the class.

**comparison operators** Operators used to find column values in a data set greater than, less than, or equal to a given value. Comparison operators define the data set query criteria.

**cross table** A cross table summarizes database information from two columns in a data set and displays it in a grid format. A cross table presents the requested information in a clear and concise format. A cross table does not change the information retrieved by a data set.

**data abstraction** Disassociating data from a specific object. An ambiguous reference.

**data feed** A Real Time data source.

**data hiding** Preventing or limiting access to object information.

**data set** An SQL query on one or more database tables.

**display map** A macro or method to manipulate information retrieved from a database. A display map changes the retrieved data value or object characteristics.

**ELF** (Extension Language Facility). The simple, yet versatile, programming language consisting of built-in macros.

**ELF directive** An element of an object method source code that indicates a special condition to the compiler. The `@@@ OBJECTS` ELF directive indicates to the compiler that the file contains Builder object source code.

---

**entry box** A text entry area dialog box control.

**event** User-defined operations called by an application when an action occurs. Each object has associated events. Applix Builder applications are event-driven; clicking on a button or typing in an entry box initiates a course of action.

**expression** User-defined columns for a database query. An expression consists of information from the table columns of a query and simple mathematical functions or aggregate SQL functions. An expression displays attributes of database information without changing the information.

**external class** A created class defining attributes you can use in multiple applications. An external class is defined in a separate file. An application can link to an external class, keeping the class separate, or localize the class, making it part of the application.

**external resources** ELF arrays, bitmaps, ELF macros, or other information that is not internal to your application. Information from other sources can be added to your application as a linked resource that remains separate from the application, or a localized resource saved with the application.

**header file** See *include file*.

**hierarchy** A ranking or ordering. In Applix Builder hierarchy refers to the base class structure, where a class (subclass) inherits attributes from another class (superclass). The subclass contains at least all the attributes of the superclass, and may contain additional attributes to further define specific functionality.

**include file** A file that contains information, such as format or constant definitions, useful to more than one object method source. An include file eliminates the need to re-type information common to two or more object method sources. The format for the include statement is INCLUDE "filename".

---

**inheritance** Object inheritance is the ability of an object to derive data attributes and methods from an existing class. A created object based upon an existing class has all methods and data members associated with the existing class, but is not limited to the existing class methods.

**invocation** Calling or executing an object method. The . notation indicates method invocation.

**join** Creating a logical table from two or more database tables. A join is created by connecting columns from each table that represents the same or similar data.

**keywords** A term reserved for programming instructions. A keyword cannot be used as a variable or a label in an object method source.

**label** Text in a dialog box that is not associated with a dialog box control.

**layered panel** A special instance of a dialog box panel that hides or displays groups of controls. Also known as a tab control.

**list box** A dialog box control that displays a group of items inside a box.

**literal** A literal string method invocation. An object method can be assigned to a variable, then invoked as a literal string using the variable. The ! notation indicates a literal string method invocation.

**macro** A collection of statements that perform specific operations. ELF macros contain variable declarations, conditional statements, expressions to be evaluated, calls to other macros, and error handling statements created using custom and ELF built-in macros.

**method** An operation that can only access the data members and attributes of its own object.

**method invocation** See *invocation*.

---

**object** The concrete occurrence of a class. An object has the behavior and attributes of a class, as well as an identifiable state.

**object declaration** Defining a variable as an object data type using the keyword `object`.

**object library** A file containing one or more objects, not necessarily related, that you can use in multiple applications. Use object libraries to decrease duplicate coding and create uniform objects across applications.

**object method source** The source of user-defined events and methods for an object. An object method source can be internal to an application, or defined in an external file.

**objvar declaration** Defining a local object data variable using the keyword `objvar`.

**option button** A dialog box control that offers a menu of options from which to choose.

**panel** A decorative dialog box element used to separate items in a dialog box.

**parent** An object to which another object belongs. A parent has one or more children, such a dialog box contains controls. In Applix Builder a *child* object does not inherit attributes from a parent object. The relationship of children and parent objects is based upon enclosure, not *inheritance*.

**pen** A definition of line, text, and drawing attributes. Use a pen to draw text and graphical objects upon a *canvas*.

**push button** A dialog box control, a single button which performs an action programmed in the `clicked_event` method.

**radio button** A dialog box control that contains a group of items from which only a single item can be chosen at a time. Normally, radio button groups are a collection of logically associated items.

---

**reserved words** See *keywords*.

**scale** A dialog box control that consists of a slider and sliding area used to gradually increment and decrement values. A scale can be used alone, or with an entry area to provide exact feedback on values.

**sibling** An object at the same level of the current object. A sibling and the current object share the same *parent*, such as two controls in the same dialog box.

**table** A dialog box control that displays information in columns and rows. A table can have horizontal and vertical scroll bars so that you can move through the table information.

**this** An implicit pointer to the current object. Use this to call the object methods from within the object method source.

**toggle button** A dialog box control that can either be turned on or off. A toggle button control is used when you have a single item that can either be chosen or not chosen.

**validator** A macro or method to validate information placed into a database. A validator changes the data value or object characteristics before placing the information in the database.

**watch list** In the Debugger, a list of variables or formatted variables from the object method sources. The values contained in the variables are displayed during application execution.

---

# Index

## Symbols

\_ 3-21  
!= 3-17  
\$ALWAYS 3-64  
% 3-21  
.abd 7-10  
.abo 7-9  
.am 2-29  
.cll 2-12  
.oll 8-3  
.olo 8-11  
< 3-17  
<= 3-17  
= 3-17  
> 3-17  
>= 3-17

## A

Accelerator key  
    assign 4-38  
    change 4-41  
Adding a database 3-7  
Adding classes 2-5  
Adding columns 3-34  
Adding conditions 3-15  
aggregate

function 3-38  
function count (\*) 3-40  
functions 3-34  
vector 3-38

## ALT

key 4-31

Ambiguous Column Reference 3-45-3-47

AND 3-28

ANYWARE\_HEARTBEAT\_RATE@ 10-6

ANYWARE\_REMOTE\_INTERFACE@ 10-7

## Application

as a macro 7-8  
closing 2-22  
delete 2-21  
distribution file 7-10  
quit 2-22  
rename 2-19  
Revert 2-21  
save new 2-16  
turbo file 7-9

Application structure 2-2

## Applix Builder

bitmaps 8-33  
described 1-2  
exiting 1-6  
preferences 1-6  
starting 1-5

Applix Builder Run resource 7-4

---

Applix object registry  
  bldrRegisterObjectFile environment  
  variable 9-15  
  location 9-15  
  macros 9-15  
Array elements  
  accelerator key 4-59  
  key access only 4-59  
  macro 4-58  
  name 4-58  
Arrow  
  in list box 4-33  
  in new menus 4-45  
Auto-Query 3-14  
axmain 9-13  
  running a macro 7-3  
  running an application 7-4  
  running multiple applications 7-5  
axnet 9-13  
  starting for remote objects 8-30

## B

Between 3-20  
Bitmaps 4-12  
bitmaps  
  used in Applix Builder 8-33  
Break point  
  clear 6-11  
  display variable values 6-17  
  multiple 6-9  
  set 6-9  
Browser  
  window 2-2

## C

Call stack 6-16  
canvas 4-10  
CanvasClass 10-1  
Cascading menus 4-31  
  add 4-49  
  display 4-34  
Choose Database 3-7  
Choose Tables 3-9  
Class  
  Add 2-5  
  Copy Method 2-5  
  Delete 2-9  
  Edit Methods 2-6  
  Properties 2-8  
Class Browser 2-3  
class inheritance 9-11  
class\_initialize\_event 2-7  
Close  
  application 2-22  
Column headings 3-36  
Column query 3-23  
combo Box 4-10  
CommonDlgClass 8-32  
Comparing columns 3-23  
Comparison operators 3-17  
COMPOSE CHARACTER  
  key 4-31  
Connector 1-4  
Connector dialog box 5-2  
Control ID 4-17  
Controls 4-14  
  deleting them 4-24  
CORBA  
  ! notation 9-10

---

- adding an object to Builder 9-9
- building a server 9-14
- data structures 9-12
- data types 9-12
- defined 9-2
- interface repository 9-4-9-5
- invocation mode 9-10
- methods 9-10
- object reference 9-7
- object server 9-4
- ORB 9-2, 9-4
- out and inout parameters 9-13
- programming languages 9-2
- SOR 9-4
- CORBA server
  - AxCallBuilderObjectMethod function 9-24
  - AxInitializeBuilderObject function 9-23
  - AxTerminateBuilderObject function 9-24
  - coding requirements 9-23
  - creating in Builder 9-19
- Creating table joins 3-41
- Cross table 5-19
- Customize Menu Bar 4-31
- Customizing column headings 3-36
- Cut 4-25

## D

- Data feed
  - creating 3-52
  - defined 3-52
  - template 3-54
- Data feed record 3-57
  - filter 3-61

- Data set
  - cross table 5-19
  - data source 5-5
  - defined 3-6
  - display map 5-8
  - table 5-16
  - validator 5-10
- Data set join 3-47
- Data source
  - creating 3-3
  - data set 5-5
  - deleting 3-5
  - editing 3-4
  - macro 5-4
  - none 5-4
  - Real Time 5-7
  - renaming 3-5
- Database query 3-6
- Debug
  - Call Stack 6-16
  - Print Variable 6-17
  - Start Debugging 6-18
  - Stop Debugging 6-18
- Debugger 6-2, 1-4
  - clear break point 6-11
  - components 6-3
  - Continue 6-15
  - displaying values 6-11, 6-17
  - Execute to Here 6-17
  - execution stack 6-16
  - ExpressLine 6-4
  - Next 6-15
  - Popup menus 6-5
  - running 6-14
  - selecting objects 6-8
  - set break point 6-9
  - starting 6-2

---

- Status area 6-4
- Step 6-15
- watch list 6-11
- Debugger mode 6-2
- Debugging an applet 10-7
- Delete
  - application 2-21
  - dialog box 2-22
- Deleting a class 2-9, 2-12
- Deleting conditions 3-28
- Designer 1-4
- dialog box
  - adding controls 4-14
  - adding controls with control ID 4-17
  - bitmap 4-12
  - controls 4-9
  - copy 4-5
  - create 4-4
  - guidelines 10-2
  - label 4-12
  - panel 4-13
  - positioning controls 4-15
  - size 10-2
  - spacing of controls 10-2
- dialog box controls
  - canvas 4-10
  - combo box 4-10
  - edit box 4-10
  - entry box 4-9
  - list box 4-9
  - option button 4-9
  - push button 4-9
  - radio button 4-9
  - scale 4-10
  - tab control 4-10
  - table 4-10

- toggle button 4-9
- dialog boxes 8-32
- Dismiss
  - Menu Bar Editor 4-54
- Dismiss push button 4-4
- Display map 5-8
- Distribution files
  - creating 7-10
  - installing 7-10

## **E**

- Edit
  - SQL source 3-49
- edit box 4-10
- Edit SQL Source 3-49
- Editing classes 2-6
- ELF macros 10-6
- elfapi.a 9-14
- ELF\_STACK@ 10-7
- Ellipse
  - in menus 4-33
- entry area 4-9
- Environment settings 1-6
- Examples 1-8
- Execute
  - Menu Bar Editor 4-53
- Execute to Here 6-17
- Execution stack 6-16
- Exit
  - dialog box 2-22
- Exiting the Menu Bar Editor 4-53
- Expressions 3-34
- ExpressLine
  - Browser 2-3
  - Debugger 6-4
  - options 4-45, 4-49

---

External classes 2-10

## F

Field filter 3-61

Field template 3-54

file

- delete 2-21

- exit 2-22

- new 2-14

- open 2-15

- options 2-14

- revert 2-21

- save as 2-9

File extension

- .abd 7-10

- .abo 7-9

- .am 2-29

- .cll 2-12

- .oll 8-3

- .olo 8-11

functions 3-34

## G

Group By 3-38

Grouping conditions 3-32

## H

Having 3-39

header file 9-22

Hidden menus 4-38, 4-39

- Keys menu 4-40

## I

icons

- used in Applix Builder 8-33

IDL compiler 9-14

IIOP protocol 9-2

implementation file 9-22

In 3-18

INITIALIZE\_CORBA\_SERVER@ 9-15

Install

- distribution files 7-10

Installing 1-5

Interface Definition Language 9-23

interface repository 9-4

invocation mode

- deferred synchronous 9-11

- one way 9-11

- synchronous 9-10

IOR 9-6

Is Null 3-22

## J

Join

- Data set 3-47

- non-equal 3-44

- outer 3-44

- two table 3-41

Join Conditions 3-43

## K

Keys menu 4-40

## L

Label 4-12

labeled panel 4-14

layered panel 4-10, 4-14

license

- Real Time SQL 3-63

---

Licensing 1-5  
Like 3-21  
    wildcard characters 3-21  
list Box 4-9  
Loading classes 2-10  
Logical operators 3-28, 3-32

## M

### Macro

call change 4-41  
creating for an application 7-8  
name 4-37  
run 7-2  
running with Applix Builder Run  
resource 7-6  
running with axmain 7-3  
trigger query 3-65

Macro data source 5-4

### Menu 4-30

about options 4-30  
add 4-44, 4-46, 4-49, 4-50, 4-51, 4-52,  
4-54  
add at beginning 4-43  
add at end 4-43  
add items 4-43  
add options 4-45  
add separators 4-50  
add within 4-43  
cascade 4-34  
change items 4-40  
cut items 4-52  
delete items 4-51  
display item information 4-36  
display items in Menu Bar Editor 4-  
33  
hidden 4-38, 4-39

items list box 4-33  
moving items 4-52  
name 4-37  
option name change 4-41  
option positions 4-53  
pasting items 4-52  
separators 4-31  
update menu bar 4-53

### Menu bar

array elements 4-58  
assigning pull-down menus 4-60  
cascading menu definitions 4-60  
creating via an array 4-55  
example of definition 4-61  
example with arrays 4-64  
menu option definitions 4-58  
pull-down menu definitions 4-59  
saving in ELF data file 4-63  
setting display state 10-9  
setting display string 10-9

### Menu Bar Editor

change menu items 4-40  
display 4-33  
display menu items 4-33  
exit 4-53  
move through menus 4-35  
start 4-31

menubar\_.am file 10-9

### Meta

key 4-31  
ME\_APPLICATION\_DLG@ 10-7  
ME\_TOGGLE\_DEBUGGER@ 10-7-10-8

### Mnemonics

about 4-31  
define 4-38

Multiple query conditions 3-28

---

## N

### Name

- menu change 4-41
- option change 4-41

Non-equal join 3-44

## O

### Object 2-23

- add 2-23
- copy 2-27
- cut 2-26
- debugging 6-8
- delete 2-26
- paste 2-27
- properties 2-27
- remote 8-29
- save as library 8-2

### Object Library 8-2

- creating 8-2
- creating new 8-3
- editing 8-5
- saving 8-10
- using 8-11, 8-22

Object method source 2-28

### object reference

- CORBA 9-7
- stringified 9-6

object server 9-4

option button 4-9

### Options

- Catch Errors 6-15

OR 3-30

### ORB

- defined 9-4

Outer join 3-44-3-45

## P

Panel 4-13

Parentheses 3-32

PenClass 10-1

Performance Hints 3-14

### Popup menu

- Browser 2-3
- Class Browser 2-13
- Debugger 6-5

### Positions

- Menu Bar vs. ExpressLine 4-53

Preferences 1-6

PRINT\_DATUM@ 10-7

push button 4-9

## Q

### Query

- auto-query 3-14
- choose tables 3-9
- conditions 3-15
- create columns 3-35
- group by 3-38
- having 3-39
- query 3-15
- range 3-25

## R

radio button 4-9

Range query 3-25

### Real Time

- table 5-17

Real Time data feed 3-52

Real Time record 3-57

---

- filter 3-61
- Real Time SQL
  - license 3-63
  - trigger query 3-64
  - trigger query macro 3-65
- Real Time template 3-54
- registering Builder objects 9-14
- REGISTER\_CALL\_BACK\_OBJECT@ 9-18
- REGISTER\_OBJECT@ 9-16
- Remote objects 8-29
  - example 8-31
  - methods 8-31
- Renaming an application 2-15
- Resources 8-15
  - adding 8-15
  - deleting 8-16
- Reusable objects 8-2
- Reusing classes 2-10
- RTS
  - trigger query 3-64
  - trigger query macro 3-65
- Run 1-4
  - Quit Application 6-19
  - Run Application 6-14
- Running a macro 7-2
- Running an application 7-2
  - with axmain 7-4
- Running multiple applications 7-5

**S**

- Sample applications 1-8
- Saving classes 2-9
- scale 4-10
- Selecting a Database 3-6
- Selecting a Table 3-9

- Separators 4-31
  - add 4-50
- server
  - maintaining a connection 10-6
- Setting query conditions 3-15
- SOR 9-4
- Source 1-4
- SQL functions 3-34
- SQL query 3-6
- SQL source code 3-49
- Starting Applix Builder
  - from Applixware 1-5
- stringified object reference 9-4, 9-6
- Subselect query 3-26
- Sybase multiple database selection 3-12

## T

- tab 4-10
- tab control 4-10
- table 4-10
- Table
  - connecting a data set 5-16
  - connecting a Real Time data source 5-17
- Temporary break point 6-17
- timestamp
  - in trigger query 3-64
- toggle button 4-9
- Tools
  - overview 1-3
- Total status area 3-11
- trigger query 3-64
- Two table join 3-41

---

## U

UNREGISTER\_OBJECT@ 9-17  
User environment settings 1-6  
User-defined class 2-5

## V

Validator 5-10  
vector aggregate 3-38  
View  
    Classes 2-3  
    headings 3-36

## W

Watch list 6-11  
    using 6-12  
Wildcard characters in Like query 3-21  
WRITE\_DATA\_FILE@ 4-63

---