Application Extension Facility (July, 1997)

Copyright Notice

Copyright © 1991, 1997 by Tivoli Systems, an IBM Company, including this documentation and all software. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of Tivoli Systems. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Note to U.S. Government Users—Documentation related to restricted rights—Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Trademarks

The following product names are trademarks of Tivoli Systems or IBM Corporation: AIX, IBM, OS/2, RISC System/6000, Tivoli Management Environment, and TME 10.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names mentioned in this document may be trademarks or servicemarks of others.

Notice

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to Tivoli System’s or IBM’s valid intellectual property or other legally protectable right, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user.

Patents may be pending.
Preface................................................................................................................................................. xiii

Chapter 1—Installing TME 10 AEF
Software Requirements ................................................................................................. 1–1
Hardware Requirements .............................................................................................. 1–1
  Clients .......................................................................................................................... 1–2
  Servers ....................................................................................................................... 1–2
Installing TME 10 AEF ................................................................................................. 1–3
  Desktop ....................................................................................................................... 1–3
  Command Line ......................................................................................................... 1–8

Chapter 2—Getting Started with TME 10 AEF
Introduction ................................................................................................................... 2–1
What TME 10 AEF Can Do ........................................................................................ 2–4
Anatomy of a Customization ...................................................................................... 2–5

Chapter 3—TME 10 Dialogs
Introduction ................................................................................................................... 3–1
TME 10 Dialogs .......................................................................................................... 3–3
  Command and Method Invocation ............................................................................ 3–3
TME 10 Dialog States ................................................................................................. 3–5
TME 10 Dialog Descriptors ....................................................................................... 3–6
TME 10 Gadgets ........................................................................................................... 3–7
  Icon Gadgets ............................................................................................................ 3–8
  Gadget Attributes ................................................................................................... 3–9
Callbacks ..................................................................................................................... 3–9
Chapter 4—Dialog Specification Language

Introduction.........................................................................................................4–1
Logical Organization of a Dialog Specification.............................................4–5
  The Main Dialog Block..............................................................................4–6
  Variable Specification................................................................................4–7
  Attribute Specification ...............................................................................4–8
Gadget Specification........................................................................................4–14
  Active Gadgets ......................................................................................4–15
  Inactive Gadgets ....................................................................................4–16
  Gadget Paths ............................................................................................4–17
Lexical Components of DSL .........................................................................4–19
  Blocks.......................................................................................................4–19
  Comments ................................................................................................4–20
  Special Characters....................................................................................4–20
  Identifiers .................................................................................................4–21
  Strings ......................................................................................................4–21
Dialog Callbacks.............................................................................................4–21
  Method Callbacks....................................................................................4–22
  Desktop Command Callbacks ...............................................................4–23
  Gadget Library Callbacks ........................................................................4–25
    Callbacks for All Gadgets .................................................................4–26
    Button Gadget Callbacks.....................................................................4–28
    Choice and List Gadget Callbacks......................................................4–28
    Group Gadget Callbacks....................................................................4–28
    Switch Gadget Callbacks....................................................................4–29
    Table Gadget Callbacks......................................................................4–29
    Dialog Callbacks ..................................................................................4–30
  Managing Callback Errors .......................................................................4–31
Bitmap Support in DSL ................................................................................4–31
Color Support in DSL ..................................................................................4–32
Internationalization Support in DSL............................................................4–40
Chapter 5—Custom Dialogs

Introduction ........................................................................................................ 5–1
Retrieving a Dialog Descriptor ........................................................................ 5–3
  Getting a Resource-Wide Dialog ................................................................. 5–4
  Getting a Per-Instance Custom Dialog ..................................................... 5–5
  Getting a Create Dialog ............................................................................ 5–5
Reverse Compiling a Descriptor .................................................................... 5–6
Installing a Custom Dialog .......................................................................... 5–7
  Customizing a Resource Type ..................................................................... 5–7
  Customizing an Instance ........................................................................... 5–8
  Customizing a Create Dialog .................................................................... 5–9
Removing a Custom Dialog .......................................................................... 5–9
  Removing a Resource-Wide Custom Dialog ............................................ 5–10
  Removing a Per-Instance Custom Dialog ................................................. 5–11
  Removing a Custom Create Dialog ............................................................ 5–11
Directory Services for Dialogs ...................................................................... 5–12
Customizing a Pull-Down Menu .................................................................... 5–13
Chapter 6—Custom Attributes

Adding a New Attribute........................................................................................................6–1
Getting the Value of an Attribute .........................................................................................6–2
Setting the Value of an Attribute .........................................................................................6–3
Removing an Attribute..........................................................................................................6–3

Chapter 7—Custom Callback Methods

Introduction.............................................................................................................................7–1
Custom Method Installation .................................................................................................7–4
  Customizing an Entire Resource Type .............................................................................7–4
  Installing a Method for All Platform Types .................................................................7–5
  Installing Platform-Specific Methods .............................................................................7–6
  Defining ACLs for a Method .........................................................................................7–8
  Setting the Privileges on a Method ..............................................................................7–9
  Customizing a Remote TMR .........................................................................................7–9
  Customizing an Instance ...............................................................................................7–10
  Installing a Method for a Create Dialog ......................................................................7–11
An Example ............................................................................................................................7–12
Removing a Custom Method...............................................................................................7–14
  Removing a Method from an Entire Resource Type .....................................................7–14
  Removing Platform-Specific Methods ..........................................................................7–14
  Removing Methods in Remote TMRs ..........................................................................7–15
Removing a Method from an Instance ................................................................................7–15
Removing a Create Dialog Method ...................................................................................7–16
Callback Method Utilities......................................................................................................7–17
  Confirmation Dialogs ......................................................................................................7–17
  Error and Message Dialogs ............................................................................................7–19
  Other Types of Dialogs ....................................................................................................7–19
  Run-Time Gadget Values ...............................................................................................7–21
  Status Window ..............................................................................................................7–25
Chapter 8—Custom Policy

Adding a Policy Method .................................................................................... 8–2
Removing a Policy Method ............................................................................... 8–3
Using Policy in a Custom Method ..................................................................... 8–4

Chapter 9—Bitmaps and Icons

XPM Bitmap Format.......................................................................................... 9–2
Bitmaps .............................................................................................................. 9–4
  Getting a Bitmap ....................................................................................... 9–6
    Getting a Bitmap Stored on a Resource.................................................. 9–6
    Getting a Per-Instance Custom Bitmap .................................................. 9–7
    Getting a Bitmap From a Create Dialog ................................................ 9–7
  Installing a Custom Bitmap ........................................................................ 9–8
    Installing a Resource-Wide Bitmap ....................................................... 9–9
    Installing a Bitmap On an Instance ....................................................... 9–10
    Installing a Bitmap For a Create Dialog ............................................ 9–10
  Removing a Custom Bitmap ...................................................................... 9–11
    Removing a Resource-Wide Bitmap ....................................................... 9–12
    Removing a Bitmap From an Instance .................................................. 9–13
    Removing a Bitmap From a Create Dialog ............................................ 9–13
  Getting a List of Bitmaps for a Resource ................................................... 9–14
Icons ................................................................................................................. 9–15
  Getting Icon Dialogs and Bitmaps ............................................................. 9–19
  Customizing an Icon ................................................................................ 9–20
  Installing an Icon ..................................................................................... 9–21
    Creating a New Icon State .................................................................... 9–21
    Replacing an Icon Bitmap .................................................................... 9–22
    Replacing an Icon Dialog .................................................................... 9–22
  Using the New Icon ................................................................................ 9–23
Examples ........................................................................................................ 9–24
  Customizing the Icon Bitmap .................................................................. 9–24
Customizing the Icon Menu ............................................................9–26
Removing a Custom Icon.........................................................................9–28
Getting Lists of Custom Icons.................................................................9–29

Chapter 10—TME 10 AEF and Profiles

Profile Properties ..............................................................................................10–2
   Adding a Profile Property ........................................................................10–2
   Removing a Profile Property....................................................................10–4
Actions ..............................................................................................................10–4
   Adding an Action .....................................................................................10–5
   Getting an Action Definition ..................................................................10–6
   Getting a List of Actions .........................................................................10–7
   Getting a List of Pending Actions ............................................................10–8
   Removing an Action ................................................................................10–9
Profile Dialogs ..................................................................................................10–9

Chapter 11—Putting It Together

Example: Adding an Attribute.......................................................................... 11-1
   Dialog Gadget .......................................................................................... 11-2
   Callback Methods..................................................................................... 11-6
Example: Adding Menu Options.................................................................... 11-9
   Pop-Up Menu ........................................................................................ 11-10
   Pull-Down Menu ................................................................................... 11-12
   Creating Callback Methods.................................................................... 11-16
      Reboot Method .................................................................................. 11-17
      Shutdown Method ........................................................................... 11-18
   Installing the Custom Methods .............................................................. 11-21
   Policy Support ....................................................................................... 11-22

Appendix A—Dialog Descriptors

Dialog Descriptors .............................................................................................A–1
   Administrator .......................................................................................... A–1
AdministratorCollection ................................................................. A–2
BulletinBoard ..................................................................................... A–3
Job ..................................................................................................... A–3
ManagedNode ..................................................................................... A–3
PCManagedNode .............................................................................. A–4
PolicyRegion ...................................................................................... A–5
ProfileManager .................................................................................. A–5
Scheduler .......................................................................................... A–6
TaskLibrary ......................................................................................... A–7
Icon States .......................................................................................... A–7
Administrator ..................................................................................... A–8
AdministratorCollection ................................................................. A–8
BulletinBoard ..................................................................................... A–8
Job ..................................................................................................... A–8
ManagedNode ..................................................................................... A–9
PCManagedNode .............................................................................. A–10
PolicyRegion ..................................................................................... A–10
ProfileManager .................................................................................. A–10
Scheduler .......................................................................................... A–11
TaskLibrary ......................................................................................... A–11

Appendix B—CLI Reference

dsl ................................................................................................. B–2
rdsl ................................................................................................. B–4
waddaction ..................................................................................... B–5
waddattr ......................................................................................... B–7
waddpolm ....................................................................................... B–8
waddprop ....................................................................................... B–10
wconbitmap ................................................................................... B–11
wdispconf ..................................................................................... B–13
wdisperr ....................................................................................... B–14
Appendix C—DSL Gadget/Attribute Reference
Accelerator ......................................................................................................... C-2
Activate .............................................................................................................. C-3
Alignment .......................................................................................................... C-4
AppendText ........................................................................................................ C-6
Background ........................................................................................................ C-7
Bitmap ................................................................................................................ C-8
BitmapTitle ........................................................................................................ C-9
Border .............................................................................................................. C-10
Button ............................................................................................................... C-11
CellStyle .......................................................................................................... C-12
Changed ........................................................................................................... C-13
ChildColumnAlignment ................................................................................... C-14
ChildRowAlignment ........................................................................................ C-16
Choice .............................................................................................................. C-18
Choices ............................................................................................................. C-20
Collection ......................................................................................................... C-21
ColumnSelection .............................................................................................. C-23
ColumnWidth ................................................................................................... C-24
Columns ........................................................................................................... C-25
CommandButton .............................................................................................. C-26
Command Dialog ............................................................................................. C-27
Commands ....................................................................................................... C-30
CopyIn .............................................................................................................. C-31
Default .............................................................................................................. C-33
DialogStyle ...................................................................................................... C-34
DoubleClick ..................................................................................................... C-35
Encrypt ............................................................................................................. C-36
Font .................................................................................................................. C-38
Foreground ....................................................................................................... C-39
Glyph ................................................................................................................ C-41
GridHeight ....................................................................................................... C-42
GridHorizontal .............................................................. C-43
GridVertical ............................................................... C-44
GridWidth ................................................................. C-45
Group ....................................................................... C-46
HelpMenu ................................................................. C-48
HelpMessage ............................................................. C-49
Icon ......................................................................... C-50
Layout ...................................................................... C-52
LayoutPolicy ............................................................. C-53
LeftColumn ............................................................... C-54
List ........................................................................... C-55
Menu ...................................................................... C-57
Message ................................................................. C-59
Mnemonic ............................................................... C-60
MoveIn .................................................................... C-61
MoveOut ................................................................. C-63
Name ..................................................................... C-65
Object ..................................................................... C-66
Off ......................................................................... C-67
On ......................................................................... C-68
Open ..................................................................... C-69
Page ...................................................................... C-70
PopDown ............................................................... C-72
Purpose ................................................................. C-73
ReadOnly ............................................................... C-74
Refresh ................................................................. C-75
RowHeight ............................................................. C-76
RowSelection ......................................................... C-77
Rows ................................................................. C-78
Scrolling .............................................................. C-79
SelectionMode ...................................................... C-80
Sensitive ................................................................. C-81
Separator .............................................................. C-82
Show ................................................................. C-83
ShowBrowser ..................................................... C-84
ShowIndicator ..................................................... C-85
Sort ................................................................. C-86
StatusField ......................................................... C-87
StatusMessage ................................................... C-88
Switch .............................................................. C-89
Table ............................................................... C-91
TableColumn ...................................................... C-94
TableRow .......................................................... C-96
TearOff .............................................................. C-98
TellModify ........................................................ C-99
Text ................................................................. C-100
Title ................................................................. C-102
TitlePos ............................................................ C-103
TopRow ............................................................ C-104
Type ............................................................... C-105
Value .............................................................. C-106
ViewBy ............................................................ C-108
Visible ............................................................. C-109
Wrap ............................................................... C-110
dsl_colors ......................................................... C-111
gadgets ........................................................ C-119
Preface

The Tivoli Management Environment™ 10 (TME 10) is a software environment, comprised of a suite of integrated software products, specifically for distributed system management. TME 10 brings coherence and the quality control found in mainframe systems to the challenge of managing multiple operating systems, network services, diverse system tasks, and constantly changing nodes and users. TME 10 encourages a team approach to distributed system management; senior system administrators can focus on analysis, strategy, and security issues, leaving the routine and daily tasks to the staff administrators.

The Tivoli Application Extension Facility (TME 10 AEF) allows you to customize and extend the capability of existing TME 10 applications. With TME 10 AEF, you can:

- Adding fields to a dialog.
- Creating custom attributes and methods for application resources.
- Creating custom icons and bitmaps.

Although TME 10 AEF is a powerful tool for extending capabilities, it cannot make fundamental changes to existing application functionality. To significantly modify the behavior of an application, you must use the TME 10 Advanced Development Environment (TME 10 ADE).

Who Should Read This Manual

This guide is intended for system administrators who wish to extend the functionality of existing applications, such as TME 10 User Administration. This book assumes that you are familiar with:

- The TME 10 Framework
- The UNIX operation system
- Shell programming
- Display environments, such as Motif
- The applications you want to customize
Related Documents

The TME 10 Framework User’s Guide presents concepts related to profile-based applications. You should be familiar with the concepts in the Framework User’s Guide before attempting to customize a profile-based application.

What This Guide Contains

Here are the topics covered in this book:

- Chapter 1, “Installing TME 10 AEF”
  Tells how to install the product.

- Chapter 2, “Getting Started with TME 10 AEF”
  Provides a brief introduction to TME 10 AEF and the TME 10 object system.

- Chapter 3, “TME 10 Dialogs”
  Discusses the components of a TME 10 dialog.

- Chapter 4, “Dialog Specification Language”
  Primer for the Dialog Specification Language (DSL), which is the language used to create TME 10 dialogs.

- Chapter 5, “Custom Dialogs”
  Contains a full discussion of how to create a custom dialog or pull-down menu.

- Chapter 6, “Custom Attributes”
  Discusses how to add attributes to a resource type or to instances of a resource.

- Chapter 7, “Custom Callback Methods”
  Covers creation and installation of callback methods.

- Chapter 8, “Custom Policy”
  Discusses how to create validation policy for your customizations.

- Chapter 9, “Bitmaps and Icons”
  Shows you how to create custom bitmaps and icons for TME 10 resources.
Preface

- Chapter 10, “TME 10 AEF and Profiles”
  Shows you how to customize profile-based applications.
- Chapter 11, “Putting It Together”
  Provides examples to show how to add attributes or menu options.
- Appendix A, “Dialog Descriptors”
  Contains a complete list of the dialog descriptors and icon states associated with resource in the TME 10 Management Framework.
- Appendix B, “CLI Reference”
  Contains manual pages for all TME 10 AEF CLI commands.
- Appendix C, “DSL Gadget/Attribute Reference”
  Contains manual pages for all DSL gadgets and attributes.

Typeface Conventions

The manual uses several typeface conventions for special terms and actions. These conventions have the following meaning:

**Bold** Commands, keywords, file names, or other information that you must use literally appear in **bold**. Data types and structure field names are also shown in **bold**.

*Italics* Variables and values that you must provide appear in *italics*.

**Bold Italic** New terms appear in **bold italic** the first time they are used.

Monospace Code examples appear in a monospace font. Operation and function signatures are also shown in monospace.

Contacting Tivoli Customer Support

We are very interested in hearing from you about your experience with the products in TME 10. We welcome your suggestions for improvements.

If you encounter difficulties with any TME 10 product, please contact your customer support representative. Tivoli also includes the
wsupport command. This command prompts you for problem information, which can be E-mailed to your support provider or saved to a text file. You can then print the saved file, and fax the resulting TME 10 Problem Report form to your support provider.
Installing TME 10 AEF

This chapter provides the information you need to install TME 10 AEF in your TME environment.

Before attempting to install this application, make certain you review the following sections.

- Software Requirements
- Hardware Requirements

Software Requirements

To do an upgrade installation using the Install Patch... facility, you must have a TME 10 AEF 3.1 installation. If you are running TME 10 AEF 3.0, first upgrade to TME 10 AEF 3.1, then to 3.2.

If you do not have TME 10 AEF installed, you must do a full installation using Install Product.... option.

For instructions on how to install TME 10 AEF, see “Installing TME 10 AEF” on page 1-3. For instructions on how to upgrade TME 10 AEF, see the release notes.

Hardware Requirements

The following tables provide the client and server disk space requirements for the TME 10 AEF application. This space is in addition to the space requirements for the management platform.
## Hardware Requirements

### Clients

<table>
<thead>
<tr>
<th>Platform</th>
<th>Libraries</th>
<th>Binaries</th>
<th>Database</th>
<th>Man Pages</th>
<th>Message Catalogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>NA</td>
<td>2 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>AT&amp;T</td>
<td>NA</td>
<td>4 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>DG/UX</td>
<td>NA</td>
<td>5 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>HP-UX</td>
<td>NA</td>
<td>3 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>Motorola</td>
<td>NA</td>
<td>4 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>Solaris</td>
<td>NA</td>
<td>2 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>SunOS</td>
<td>NA</td>
<td>4 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Servers

<table>
<thead>
<tr>
<th>Platform</th>
<th>Libraries</th>
<th>Binaries</th>
<th>Database</th>
<th>Man Pages</th>
<th>Message Catalogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>NA</td>
<td>2 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>AT&amp;T</td>
<td>NA</td>
<td>4 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>DG/UX</td>
<td>NA</td>
<td>5 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>HP-UX</td>
<td>NA</td>
<td>3 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>Motorola</td>
<td>NA</td>
<td>4 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>Solaris</td>
<td>NA</td>
<td>2 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
<tr>
<td>SunOS</td>
<td>NA</td>
<td>4 MB</td>
<td>NA</td>
<td>1 MB</td>
<td>NA</td>
</tr>
</tbody>
</table>

1-2  
Version 3.2
Installing TME 10 AEF

You can install the TME 10 AEF application from the desktop or command line.

Desktop

Use the following steps to install the application from the TME 10 desktop. You must have the senior authorization role to install this application.

1. Select the Install->Install Product... option from the Desktop menu to display the Install Product dialog.

If the Select Product for Install: scrolling list is empty, proceed to step 2. If there are products listed in the scrolling list, skip to step 3.
2. Press the **Select Media...** button to display the **Set Media Path** dialog.

The **File Browser** dialog enables you to identify or specify the path to the installation media.

If you already know the path to the CD-ROM image

a. Enter the full path in the **Path Name** field.

b. Press the **Set Path** button to change to the specified directory.

c. Press the **Set Media & Close** button to save the new media path and return to the **Install Product** dialog. The dialog now contains a list of products that are available for installation.

If you do not know the exact path to the CD-ROM image

a. From the **Host** scrolling list, choose the host on which the install media is mounted. Choosing a host updates the **Directories** scrolling list to show the directories of the host you chose.
b. From the Directories scrolling list, choose the directory containing the install media. Choosing a directory updates the Files scrolling list to show the files contained in the directory you chose.

- If you wish, enter a regular expression in the File Name Filter field to limit the number of file names in the Files scrolling list. See Chapter 2 of the TME 10 Framework User’s Guide for information on regular expressions.

- Press Filter to update the Files scrolling list with only those files matching the regular expression. You can then select the correct file.

c. Choose a file from the Files scrolling list.

d. Press the Set Media & Close button to save the new media path and return to the Install Product dialog. The dialog now contains a list of products that are available for installation.

3. Select Application Extension Facility from the Select Product to Install: list.
4. Use arrow buttons to move the clients from one choice list to another. The product will be installed on the clients in the **Clients to Install On** list.

5. Enter your license key for this product in the **License Key** field.

6. Press the **Select Install Options...** button to view and set any special installation parameters required by product. TME 10 AEF does not require any special installation options, so this button is inactive.

7. Press the **Install & Close** button to install the product and close the **Install Product** dialog.

    — OR —

    Press the **Install** button to install the product and keep the **Install Product** dialog open. You can then install the same product on another set of clients or you can install another product.
The installation process prompts you with a **Product Install** dialog similar to the following. This dialog provides the list of operations that will take place when installing the software. This dialog also warns you of any problems that you might want to correct before you install the product.

8. Press the **Continue Install** button to display the **Product Install** status dialog. This dialog displays status information as the installation process proceeds.
Installing TME 10 AEF

When the installation is the complete, the **Product Install** dialog returns a completion message similar to the one below.

9. Press the **Close** button to close the dialog.

**Command Line**

You can use the `winstall` command to install TME 10 AEF from the command line.

```
winstall -c cdrom_path -i catalog [-l license_key] [install_path...]
```

where:

- `-c cdrom_path` Specifications the complete path to the cdrom image.
-i catalog Specifies the path to the TME 10 file used to index the CD-ROM image.

-1 license_key Specifies the license key specific to the TME 10 region and TME 10 product.

install_path Path overrides of default install directories. The following are valid arguments:

**BIN** binaries_directory
  Overrides the default installation path for the TME 10 Framework binaries.

**LIB** libraries_directory
  Overrides the default installation path for the TME 10 Framework libraries.

**MAN** man_page_directory
  Overrides the default installation path for the TME 10 Framework manual pages.

**CAT** message_catalog_directory
  Overrides the default installation path for the TME 10 Framework message catalogs.
Getting Started with TME 10 AEF

Introduction

Before you attempt to customize TME 10 applications you should understand something about how TME 10 and its applications work. This chapter introduces concepts basic to the operation of TME 10 and its applications; you should read and understand this chapter before reading subsequent chapters in this guide.

TME 10 and its applications are object-oriented systems. In an object-oriented system, data and the procedures that manipulate it are treated as one entity, called an object. In TME 10, each object represents a real-world or logical system resource (user, managed node, printer, etc.).

Each object has a specific type, called its resource type. A resource type defines the attributes supported by all objects of that type. Attributes are the data that define an object’s characteristics, similar to the fields of a database record. For example, the Employee resource type might define the following attributes:

- Name
- Address
- Phone_Number
- Vacation_Accrued
- Salary
All *instances* of the *Employee* type have the above attributes, although the values assigned to them differ from object to object. In this way, each employee in a company can be represented by an instance of the *Employee* resource type.

In addition to attributes, a resource type also defines the following:

- Methods supported by instances of the type
- Dialogs and bitmaps that comprise the resource’s user interface
- Icons that represent instances of resource on the Desktop

A method is a function that manipulates an instance’s attributes. For example, an *Employee* type might have a `set_salary` method, which sets the value of the instance’s *Salary* attribute.

A resource type can be conceptually divided into three components: behavior, user interface, and extension. Look at the figure below.

<table>
<thead>
<tr>
<th>Component</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Behavior</td>
<td><em>method1</em></td>
</tr>
<tr>
<td></td>
<td><em>method2</em></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Resource User Interface</td>
<td><em>dialog1</em></td>
</tr>
<tr>
<td></td>
<td><em>bitmap1</em></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Resource Extensions</td>
<td><em>cust_method1</em></td>
</tr>
<tr>
<td></td>
<td><em>cust_dialog1</em></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
A resource’s behavior component contains the application methods supported by instances of the resource. The user interface component contains the application dialogs and bitmaps supported by instances of a resource. Initially, all instances of a resource support the same set of methods, dialogs, and bitmaps.

Each TME 10 resource also provides a special component to facilitate extensions and local customizations. This component, initially empty, is where the modifications you make with TME 10 AEF are stored. For example, suppose you create a custom version of the Managed Node Properties dialog. TME 10 AEF stores the custom version in the extension component of the ManagedNode resource type. When a user opens the Properties dialog, TME 10 automatically uses the custom version. Since the custom version does not overwrite the original version, you can simply remove the custom version to revert to the original.

Modifications stored in a resource’s extension component affect all instances of the resource. TME 10 AEF also gives you the ability to customize single instances of a resource. In addition to the extension component on the resource type, all instances of a resource have an extension component.
Customizations stored in an instance’s extension component affect only that instance.

### What TME 10 AEF Can Do

The table below summarizes the capabilities of TME 10 AEF.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Can TME 10 AEF?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new gadgets to existing dialogs</td>
<td>Yes</td>
</tr>
<tr>
<td>Make gadgets in a dialog insensitive</td>
<td>Yes</td>
</tr>
<tr>
<td>Add items to a pop-up or pull-down menu</td>
<td>Yes</td>
</tr>
<tr>
<td>Remove items from pop-up or pull-down menus</td>
<td>Yes</td>
</tr>
<tr>
<td>Change dialog text (titles, labels, and so on)</td>
<td>Yes</td>
</tr>
<tr>
<td>Add custom attributes to a resource type</td>
<td>Yes</td>
</tr>
<tr>
<td>Remove existing attributes from a resource type</td>
<td>No</td>
</tr>
<tr>
<td>Add custom methods to a resource type</td>
<td>Yes</td>
</tr>
<tr>
<td>Remove existing methods from a resource type</td>
<td>No</td>
</tr>
<tr>
<td>Create custom icons</td>
<td>Yes</td>
</tr>
<tr>
<td>Add new properties to a profile-based application</td>
<td>Yes</td>
</tr>
<tr>
<td>Create new resource types</td>
<td>No</td>
</tr>
<tr>
<td>Create new dialogs</td>
<td>No</td>
</tr>
<tr>
<td>Add/Modify Choice and List gadgets</td>
<td>No</td>
</tr>
<tr>
<td>Remove non-custom items from a dialog</td>
<td>No</td>
</tr>
<tr>
<td>Create custom dialogs for each administrator</td>
<td>No</td>
</tr>
<tr>
<td>Modify drag-and-drop operations</td>
<td>No</td>
</tr>
</tbody>
</table>

The items listed as “No” in the table above require more direct access to the TME 10 Framework. If you want to perform any of these
operations, you must use the TME 10 Advanced Development Environment (TME 10 ADE).

Anatomy of a Customization

Many of the customizations done with TME 10 AEF involve adding a gadget to a dialog or a command to a menu. This involves the following:

■ Creating a custom version of a dialog that contains a new gadget or field.
■ Installing the new dialog on a resource or instances of a resource.
■ Adding attributes to the resource to store the value of the new gadget.
■ Adding callback methods to act as the link between the custom gadgets and the new attributes.
■ Defining validation policy for the new attributes.

In addition, you can use TME 10 AEF to define custom bitmaps for a resource’s dialogs or customize the icons that represent a resource.
Anatomy of a Customization
Introduction

An intuitive graphical user interface enhances the usability of any application. A well-designed interface displays information about the application, such as resources and actions that can be performed on those resources.

The TME 10 user interface is called a *desktop*. Each authorized TME 10 administrator has access to a TME 10 desktop. The TME 10
Introduction

desktop displays a dialog containing one or more icons each of which represent system resources.

When expanded, the icon is replaced by a dialog that allows administrators to communicate with the associated resource. These resource dialogs enable administrators to change system configurations and create new resources in the distributed environment.

When an administrator selects a gadget or initiates an operation using drag-and-drop, the user interface invokes one or more methods (known as callbacks) on an object associated with the resource.

Although it is not apparent to the administrator, the callbacks might not be directly invoked on the resource objects or even run on the local
machine. Instead, they are translated into a series of method
invocations, which perform the work on one or more objects
distributed across the network. They then return any results or status
to the administrator from these objects.

Integral to an understanding of TME 10 AEF is a knowledge of TME
10 dialogs. A TME 10 dialog is a window that contains controls for
communicating with a TME 10 application.

TME 10 Dialogs

Each TME 10 dialog serves as a two-way communication link between
an administrator and one or more TME 10 resources. TME 10
resources work with dialog descriptors, which are data structures that
define dialog contents and behavior. Dialog descriptors are specified
using the Dialog Specification Language (DSL), which is provided as
a part of TME 10.

Dialogs can exhibit two basic kinds of behavior: command and method
invocation, and drag-and-drop.

Command and Method Invocation

When you specify dialogs using DSL, you specify (among other
things) the gadgets it contains. Some gadgets allow the administrator
to send messages from the dialog to the desktop object, which then
invokes an internal command or an external method.
Dialogs that contain such gadgets typically have one or more fields for entering data, and one or more command buttons as shown in the following example:

The example dialog contains three types of component for communication with the application:

- A text field that displays a current alias list for the node
- A text field for entering new aliases
- Four command buttons for passing instructions to the TME 10 desktop:
  - Add
  - Addresses
  - OK
  - Cancel

Each of the command buttons displayed in the example dialog represents an operation that is performed when the administrator selects it. The TME 10 desktop can either internally perform the operation, or it can initiate a request to an external method. The action
taken by the TME 10 desktop is determined by the method given in the
DSL dialog specification.

DSL specifies not only the dialog contents (attributes), but its behavior
as well. The behavior of the dialog is specified in the form of
callbacks—commands or methods that are invoked when a gadget is
selected.

For each command button there should be a specification of callbacks
that initiate the required operation. It is the callback specification that
determines whether the operation is executed as a desktop command
or a method.

TME 10 Dialog States

Dialogs exist in a set condition (state) that determines their behavior.
Each dialog is always in one of the following three states:

- **visible**—The TME 10 desktop has been notified of the dialog’s
  existence and it displays the dialog on the Administrator’s
  Desktop. If the dialog has active components, the administrator
  can interact with it.

- **defined but invisible**—The TME 10 desktop has been notified of
  the dialog’s existence, but does not display it. The dialog is not
  present as a window, nor is it present as an icon; the administrator
  cannot interact with it.

- **undefined**—The TME 10 desktop has not been notified of the
  dialog’s existence and it does not display it.

When it is necessary, you can change dialog states. The transitions
among these states follow:

- **Posting** an undefined dialog makes it defined but invisible.

- **Presenting** a defined but invisible (posted) dialog makes it
  visible.

- **Dismissing** a visible dialog returns it to a defined yet invisible
  state.

- **Freeing** a dialog makes it undefined, regardless of its previous
  state.
The following diagram shows the relationship among these states:

Transition from an undefined state requires you to bind values of the appropriate kind to one or more dialog variables.

**TME 10 Dialog Descriptors**

A *dialog descriptor* is a binary file that describes dialog contents and behavior. Dialog descriptors have three primary component types:

- **Dialog attributes** — Specify global information about the dialog; for example, the dialog’s **Name** or **Title**.

- **Gadgets and their attributes** — Specify an individual unit of displayed information, one or more operations an administrator can invoke, the way the information is displayed, or a combination of these things.

- **Desktop variables** — Similar to program variables; they provide an easy way to store, access, and change local information at run-time.

These components work together to specify the following:

- Information displayed by a dialog.
- Data values displayed by a dialog instance.
- The way information is displayed.
- The commands that can be invoked.
Dialog descriptors are generated statically using DSL. DSL is a compiled language with a syntax similar to that of C. Using it, you can specify dialogs statically, then compile and install them with an application. After composing your dialogs in DSL you compile them using the DSL compiler, which produces a binary dialog descriptor. You can then install the dialog descriptor. The TME 10 desktop posts your custom dialog when the application instructs it to do so.

**TME 10 Gadgets**

Dialogs represent the medium through which administrators communicate with application resources. Administrators do not actually interact with dialogs: they interact with controls presented by the dialog. TME 10 dialog controls are called gadgets, which perform the same function as Motif widgets.

A gadget presents application data and a way for administrators to provide input to dialogs. The following example dialog depicts several gadgets:
The buttons labeled Add and Addresses are Button gadgets. The buttons labeled Ok and Cancel are special buttons called CommandButton gadgets.

**Icon Gadgets**

DSL supports several kinds of gadgets. Most present simple information or invoke operations as depicted by the example dialog. Some dialogs, however, enable administrators to manipulate groups of objects.

Such applications usually present dialogs that contain icons. Icons are gadgets, which visually represent objects that model some resource. Because they represent objects, administrators can use icons to identify objects. Such identification can include any of the following:

- They can be used to identify individual objects.
- They can be used to identify object types.
- They can be used to depict the state of an object.

Icons provide information to administrators about objects. A primary part of that information includes the identification of characteristics. Administrators must be able to associate icons with individual objects to manage those objects properly. It is therefore common to attach a label to each icon.

A typical example is an icon that represents a managed node object. In such cases, each icon has a label that contains the name of the managed node. This uniquely identifies each icon and associates it with a specific managed node object.

Administrators must also be able to identify the object type represented by an icon. For example, they must be able to determine visually whether the icon represents a resource or a user object.

If the icon represents a resource, they must also be able to determine visually which resource the icon represents. So the appearance of an icon that represents a printer object must be different than the appearance of an icon that represents a host object.

The icon appearance is defined in a bitmap. A bitmap is an image representation contained in a file. TME 10 AEF allows you to create
and edit custom icon bitmaps with the icon tools available on your system.

When an administrator manages or manipulates objects, the state of those objects often changes. For example, the appearance of a bulletin board icon changes when it receives a notice. Users of the bulletin board application can immediately identify the state of their bulletin board; they can tell if a new notice has arrived.

State identification requires a separate bitmap for each state. TME 10 loads the appropriate bitmap every time the object state changes.

Gadget Attributes

Gadget attributes describe properties that can be assigned to gadgets. For example, you might want to display a title that identifies the gadget to an administrator. Using attributes, such properties can be assigned permanently to the gadget.

Each gadget supports a unique set of attributes, and so an explanation of the entire gadget attribute set is beyond the scope of this introduction. Gadget attributes are described in Chapter 4, “Dialog Specification Language.”

Callbacks

When users request an operation from a gadget, the TME 10 desktop performs an operation known as a callback. A callback is the means by which dialogs and gadgets respond to user input.

Callbacks typically transfer data and control out of the user interface and back to underlying application objects. Callbacks specify an operation, which can take the form of an internal command or an external program.

Internal commands are also called desktop commands. Desktop commands affect only the user interface; they do not communicate with the application. External programs consist of methods invoked on application resources. For example, when a user presses a button to create a user, the dialog invokes an external callback to create an instance of the user resource type.
Callback specifications are stored as dialog and gadget attributes. The TME 10 desktop supports two types of callbacks:

- Method callbacks — Callbacks to methods outside the TME 10 desktop.
- Desktop command callbacks — Callbacks to commands within the TME 10 desktop.
- Gadget library callbacks — Callbacks to a type-safe, object-oriented API.

See Chapter 4, “Dialog Specification Language” for more information about dialog callback methods and desktop commands.
Dialog Specification Language

Introduction

Traditionally, applications that use a GUI make calls to a presentation system such as Motif. This technique is error-prone and time-consuming. Further, a GUI developed using this technique does not support multiple presentation systems.

The Dialog Specification Language (DSL) simplifies the process of designing dialogs, and eliminates any dependence on a specific presentation system. DSL defines the dialogs for TME 10 applications. You can use DSL to produce a wide variety of dialogs that vary in complexity. You can, for example, produce a simple dialog to post a message for the user. You can also produce sophisticated dialogs that help the user manage network conditions.

DSL is a compiled language with a syntax similar to that of C. Using it, you can specify dialogs statically, then compile and install them with your application.
To introduce DSL, we will look at the definition for a simple dialog.

The DSL code for this dialog is shown below.

```c
#include "mddo.h"
/*
 * Purpose: "Applying Check Policy" dialog
 * Description: creates a page dialog for the results for check policy
 * Dialog Variables:
 * Callback Methods:
 */

Command Dialog {
    /* Define variables used in this dialog. */
    Variables {
        CString PAGE_RESULTS = get_results@();
        CString REGION_LABEL = get_label@();
    }
    Attributes {
```
DSL is comprised of blocks, which describe an aspect of the dialog. A block can contain subordinate blocks. Each block begins with a keyword followed by a left brace and each ends with a right brace.

In the example specification above, the main dialog block begins on line 13. The main block always begins with the keywords **Command Dialog**.

In this example, the main block is comprised of three other blocks:

- A variable block
- An attribute block
- A gadget block
The variable block, which starts with the Variables keyword on line 16 of the example, declares variables used in the dialog.

```csharp
43 Variables {
44  CString PAGE_RESULTS = get_results@();
45  CString REGION_LABEL = get_label@();
46 }
```

DSL variables are similar to variables in other programming languages. DSL variables have names and types, they can be assigned values, and you can refer to their values by name. The Variables block in the example declares two string variables: PAGE_RESULTS and REGION_LABEL. The variable block also defines values for the variables. In the example, the variables are bound (assigned values) using two callback methods (get_results and get_label).

The attribute block, which begins with the Attributes keyword on line 21 of the example, defines the dialog’s attributes and assigns them values.

```csharp
47 Attributes {
48  Name = check_policy_results;
49  Title = $REGION_LABEL;
50 }
```

Every dialog has at least one attribute: Name. The Name attribute is used to refer to the dialog in DSL commands and statements. In this example, the dialog’s name is check_policy_results.

The example dialog also has a Title attribute, which is the value displayed at the top of the dialog. The Title attribute is assigned the value of the REGION_LABEL variable (Southwest Region). The $ symbol that precedes the variable name is used to refer to the value of a variable.

The gadget block, which begins with the Gadgets keyword on line 26 of the example, defines the gadgets that appear on the dialog.

```csharp
51 Gadgets {
52  Page {
53    Name = results;
54    Title = "Check Policy Results";
55    TitlePos = TOP;
56    Columns = 75;
57    Value = $PAGE_RESULTS;
58    Sensitive = NO;
```
Logical Organization of a Dialog Specification

The basic structure of the DSL language is the block. A block is a set of statements enclosed in curly braces ({}). Each statement in the block is terminated by a semicolon (;). The type of block is dictated by
the keyword that precedes the block. For example, a block of variables begins with the keyword `Variables`.

```c
Variables {
    CString PAGE_RESULTS = get_results0();
    CString REGION_LABEL = get_label0();
}
```

Blocks can occur many times in any order. For example, twenty `Variables` blocks, each declaring a single variable, has the same effect as one `Variables` block that declares twenty variables.

As you saw in the example dialog specification at the beginning of the chapter, DSL dialogs are specified by declaring a dialog component and then describing that component using DSL keywords. Fundamentally, a dialog specification can consist of four components:

- Main dialog block.
- Variable specification.
- Attribute specification.
- Gadget specification.

The variable, attribute, and gadgets specifications are contained within the main dialog block. Every dialog must have a main dialog block, and every dialog must have a `Name` attribute associated with it.

### The Main Dialog Block

Each DSL source file should contain the definition of a single command dialog. The main block of a DSL file defines the type of dialog, and encapsulates the dialog’s gadget and attribute definitions. In TME 10 AEF, you will work with `command dialogs`. Command dialogs are defined using the keywords `Command Dialog` before the main block.

```c
Command Dialog {
    // Dialog definition
}
```

See the `CommandDialog` manual page for details about command dialogs.
Variable Specification

Conceptually, DSL variable declaration is no different than variable declaration in C. The syntax for doing so, however, is different.

Variables for a dialog are declared in a Variables block. As with C, you must declare your variables before you use them. The example below shows a sample Variables block.

```
Variables {
    CString aux_view = update@("161", "686");
}
```

The block declares a dialog variable `aux_view` of type `CString` (a standard C string). The variable is assigned an initial value by invoking the `update` method.

DSL supports three variable data types

- **String** — Used in the same way a char * variable is used in C.
- **CString** — Represents a data type encoded as a string.
- **CMsg** — Represents a data type encoded as a message.

Use `String` when the data type will be passed to the dialog as a string. This is different from `CString` data types, which should be used when the variable represents a complex data type such one of the following:

- A callback result.
- The list of choices in a `Choice` gadget.
- Any type other than a simple string.

Use `CMsg` when the variable represents data that is passed from a message catalog using the message directive. The message catalogs and the message directive is typically used in applications that employ internationalized dialogs.

Dialog variables are **bound** (assigned a value) only once; when an instance of the dialog is posted to the Desktop.

Avoid defining dialog variables and gadget variables with the same name. The Desktop interprets references to such variable names as references to the dialog variable.
Logical Organization of a Dialog Specification

There are several variables that are defined by the system. They are listed in the table below.

<table>
<thead>
<tr>
<th>Special Variable</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$drops</td>
<td>The object references of the objects represented by the set of icons dropped onto a collection. This variable is only useful in drag rules. Callbacks that use this variable as an argument can receive multiple object-ID arguments.</td>
</tr>
<tr>
<td>$icon</td>
<td>The object reference of the object represented by an icon. Use this variable only in callback arguments.</td>
</tr>
<tr>
<td>$owner</td>
<td>The object reference of the dialog owner. This value is the default object used for method callbacks. You can use this variable anywhere a dialog variable is valid.</td>
</tr>
<tr>
<td>$selects</td>
<td>The object references of the objects represented by the selected icons in a collection. Callbacks that use this variable as an argument can receive multiple object-ID arguments. Use this variable only in callback arguments or to define the standard input to a callback. Pass this variable on the standard input.</td>
</tr>
<tr>
<td>$self</td>
<td>The instance ID of a dialog. Use this variable to post multiple instances of a single dialog.</td>
</tr>
</tbody>
</table>

Attribute Specification

A primary component of any dialog specification is the specification of dialog and gadget attributes. Attribute specifications contain one or more statements that define attributes for a Dialog or Gadgets block.

To define attributes for a dialog, use an Attributes block. The code fragment below shows an example of an Attributes block.

```plaintext
Command Dialog {
    Attributes {
        Name = Host_properties;
        Title = "Host Properties";
    }
}
```
The block defines two attributes: **Name** and **Title**. TME 10 requires every dialog to have a name, so every DSL file must contain at least contain a definition for the **Name** attribute. When defining attributes for a gadget, you can simply declare the attribute inside the gadget specification.

Attribute keywords are not reserved for a particular dialog or gadget type; they frequently specify similar properties for dialogs and several types of gadgets. The **Name** keyword, for example, can be used to specify the name of a command dialog or any gadget except a **Collection** gadget.

The table below lists the available attributes in DSL. Not all attributes are valid for every gadget type. See the appropriate manual page for a list of attributes supported by a gadget.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator</td>
<td>Specifies keyboard accelerators for the current gadget</td>
</tr>
<tr>
<td>Activate</td>
<td>Specifies a callback list to be invoked for a row or column of a table gadget; when you specify an activate callback list for a row or column, the dialog presents a button at the beginning of that row or column that executes the callback when selected</td>
</tr>
<tr>
<td>Alignment</td>
<td>Specifies the alignment of gadgets in a dialog</td>
</tr>
<tr>
<td>AppendText</td>
<td>Specifies text to be appended to the current gadget</td>
</tr>
<tr>
<td>Background</td>
<td>Specifies the background color of a dialog or gadget</td>
</tr>
<tr>
<td>BitmapTitle</td>
<td>Specifies a bitmap to be used as a dialog or gadget title; specify either the bitmap name or the name of the file that contains the bitmap (see “Bitmap Support in DSL” on page 4-31 for information about specifying a bitmap file)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Border</td>
<td>Specifies whether to draw a border around the gadgets in a Group in DSL.</td>
</tr>
<tr>
<td>Changed</td>
<td>Specifies callbacks to be invoked when an administrator finishes making changes to the value of a gadget.</td>
</tr>
<tr>
<td>ChildColumnAlignment</td>
<td>Specifies the horizontal gadget alignment for dialogs that use grid alignment rather than manual dialog layout — use only when Layout is set to Grid</td>
</tr>
<tr>
<td>ChildRowAlignment</td>
<td>Specifies the vertical gadget alignment for dialogs that use grid alignment rather than manual dialog layout — use only when Layout is set to Grid</td>
</tr>
<tr>
<td>Choices</td>
<td>Specifies a choice list.</td>
</tr>
<tr>
<td>Columns</td>
<td>Specifies the width of the current gadget by specifying the number of text columns it contains.</td>
</tr>
<tr>
<td>ColumnSelection</td>
<td>Specifies how columns in a table gadget can be selected: not at all, one at a time, or multiple columns at a time.</td>
</tr>
<tr>
<td>ColumnWidth</td>
<td>Similar to Columns; specifies the width of a table column by specifying the number of text columns it contains.</td>
</tr>
<tr>
<td>Commands</td>
<td>Specifies callbacks to be invoked when the gadget is selected.</td>
</tr>
<tr>
<td>CopyIn</td>
<td>Specifies drag rules for copying icons into a collection or onto another icon.</td>
</tr>
<tr>
<td>Default</td>
<td>Specifies a default command for dialog.</td>
</tr>
<tr>
<td>DialogStyle</td>
<td>Specifies the dialog mode.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>Specifies the callback list to be invoked when the administrator double-clicks on the current choice list or icon.</td>
</tr>
<tr>
<td>Encrypt</td>
<td>Specifies whether to encrypt the text value in Text gadgets.</td>
</tr>
<tr>
<td>Font</td>
<td>Specifies XLFD font.</td>
</tr>
<tr>
<td>Foreground</td>
<td>Specifies the background color of a dialog or gadget.</td>
</tr>
<tr>
<td>Glyph</td>
<td>Specifies a bitmap for a table row or column; similar to BitmapTitle.</td>
</tr>
<tr>
<td>GridHorizontal</td>
<td>Specifies the relative horizontal placement of a gadget—use only when Layout is set to Grid.</td>
</tr>
<tr>
<td>GridVertical</td>
<td>Specifies the relative vertical placement of a gadget—use only when Layout is set to Grid.</td>
</tr>
<tr>
<td>Help</td>
<td>Specifies callbacks for displaying on-line help.</td>
</tr>
<tr>
<td>HelpMenu</td>
<td>Specifies the name of a menu to be used as a help menu.</td>
</tr>
<tr>
<td>HelpMessage</td>
<td>Specifies a status message to be displayed when an administrator points at an icon or at a menu item.</td>
</tr>
<tr>
<td>Iconic</td>
<td>Specifies whether a dialog is initially presented as an icon.</td>
</tr>
<tr>
<td>Layout</td>
<td>Specifies whether gadgets are presented vertically, horizontally, or using a relative grid.</td>
</tr>
<tr>
<td>LayoutPolicy</td>
<td>Specifies whether to populate a table horizontally or vertically.</td>
</tr>
</tbody>
</table>
## Logical Organization of a Dialog Specification

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LeftColumn</strong></td>
<td>Specifies which table column is to be presented as the one on the extreme left of the table when it is initially presented; any columns to the left of the one specified can be viewed by scrolling leftward</td>
</tr>
<tr>
<td><strong>MoveIn</strong></td>
<td>Specifies drag rules for moving icons into a collection or onto another icon</td>
</tr>
<tr>
<td><strong>MoveOut</strong></td>
<td>Specifies drag rules for moving icons out of a collection or onto another icon</td>
</tr>
<tr>
<td><strong>Mnemonic</strong></td>
<td>Specifies a letter in a menu title that is underscored; the menu can then be selected by entering that letter from the keyboard</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Specifies the name of a dialog or gadget—names are used by your application to identify a dialog or gadget; names are different than titles, which are used by administrators to identify a dialog or gadget</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td>Specifies the object reference of an object represented by an icon</td>
</tr>
<tr>
<td><strong>Off</strong></td>
<td>Specifies the value of a switch when an switch is turned off</td>
</tr>
<tr>
<td><strong>On</strong></td>
<td>Specifies the value of a switch when an switch is turned on</td>
</tr>
<tr>
<td><strong>Open</strong></td>
<td>Specifies callbacks to be invoked when an icon is opened</td>
</tr>
<tr>
<td><strong>PopDown</strong></td>
<td>Specifies callbacks to be invoked when a dialog is removed from a display</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Specifies the purpose of a dialog</td>
</tr>
</tbody>
</table>
## Logical Organization of a Dialog Specification

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Specifies the gadget as a read-only gadget; the gadget value cannot be changed from the GUI</td>
</tr>
<tr>
<td><strong>Refresh</strong></td>
<td>Specifies callbacks for refreshing a dialog</td>
</tr>
<tr>
<td><strong>Rows</strong></td>
<td>Specifies the number of text rows to display</td>
</tr>
<tr>
<td><strong>RowSelection</strong></td>
<td>Specifies how rows in a table gadget can be selected; not at all, one at a time, or multiple columns at a time</td>
</tr>
<tr>
<td><strong>Scrolling</strong></td>
<td>Specifies whether gadget members can be displayed in a scrolling window</td>
</tr>
<tr>
<td><strong>SelectionMode</strong></td>
<td>Specifies what can be selected on a table gadget; ROW, COLUMN, BOTH, NONE</td>
</tr>
<tr>
<td><strong>Sensitive</strong></td>
<td>Specifies whether a gadget is sensitive to administrator input</td>
</tr>
<tr>
<td><strong>Show</strong></td>
<td>Specifies how many choices are to be simultaneously displayed</td>
</tr>
<tr>
<td><strong>ShowBrowser</strong></td>
<td>Specifies whether to display a browser for a choice list</td>
</tr>
<tr>
<td><strong>ShowIndicator</strong></td>
<td>Specifies whether to present an indicator for a switch to indicate whether the switch is on or off</td>
</tr>
<tr>
<td><strong>Sort</strong></td>
<td>Specifies whether choices in a choice list are to be displayed in ascending alphabetical order</td>
</tr>
<tr>
<td><strong>StatusField</strong></td>
<td>Specifies whether a command dialog has a status-message field</td>
</tr>
<tr>
<td><strong>StatusMessage</strong></td>
<td>Specifies the current status message to be displayed in a dialog</td>
</tr>
</tbody>
</table>
Gadget Specification

Gadget specifications contain one or more statements that define gadgets and gadget attributes. Most gadgets consist of the DSL name for the gadget followed by a list of attribute definitions enclosed in curly braces. These attributes can be given in any order.
The following code fragment shows a **Text** gadget specification. A **Text** gadget is a simple text entry field.

```
Gadgets {
    Text {
        Name = home_dir;
        Title = "Home Directory:";
        Value = $initial_home_dir;
    }
}
```

The **Name** and **Title** attributes set those properties of the gadget. The **Value** attribute of a gadget defines the initial contents of the gadget. In a **Text** gadget, this is the text presented for the user to edit.

Gadget keywords specify gadgets, which are the dialog controls used by administrators to request operations of your application. DSL supports several gadget types, each of which are classified as either **active** or **inactive**.

### Active Gadgets

**Active** gadgets are those that can accept user input. Active gadgets can be classified as either **command** gadgets or **value** gadgets.

Command gadgets invoke operations when they are selected. The table below list the DSL command gadgets.

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>Invokes a callback to start an operation</td>
</tr>
<tr>
<td>CommandButton</td>
<td>Button presented at the bottom of a dialog</td>
</tr>
<tr>
<td>Icon</td>
<td>Presents a visual representation of an object, and a set of operations for the object through a pop-up menu</td>
</tr>
</tbody>
</table>

Value gadgets display one or more values and allow the user to change the gadget value. Value gadgets can be used in two ways:

- To arrange for an operation to be performed every time the gadget value is changed
To send a value related to the state of the gadget to a callback. See Chapter 7, “Custom Callback Methods” for more information about callbacks.

The table below lists the DSL view gadgets.

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice</td>
<td>Presents a list of choices and allows a maximum of one entry to be selected</td>
</tr>
<tr>
<td>List</td>
<td>Presents a list of choices and allows multiple entries to be selected</td>
</tr>
<tr>
<td>Switch</td>
<td>Presents a switch with a simple on/off state</td>
</tr>
<tr>
<td>Page</td>
<td>Presents a multi-line text field</td>
</tr>
<tr>
<td>Table</td>
<td>Presents a multi-row, multi-column editable table</td>
</tr>
<tr>
<td>Text</td>
<td>Presents a one-line text string</td>
</tr>
</tbody>
</table>

**Inactive Gadgets**

Inactive gadgets can only display data or contain other gadgets. The table below lists the DSL inactive gadgets:

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>Presents a dialog sector containing a collection of member icon gadgets</td>
</tr>
<tr>
<td>Message</td>
<td>Presents a non-editable text string</td>
</tr>
<tr>
<td>Separator</td>
<td>Presents a single horizontal or vertical separating line</td>
</tr>
<tr>
<td>Group</td>
<td>Presents a set of gadgets as a group. A group gadget can contain most other gadgets, including other groups</td>
</tr>
<tr>
<td>Menu</td>
<td>Presents a menu of item gadgets</td>
</tr>
</tbody>
</table>
Gadget Paths

Each gadget is defined by a series of attributes that you specify in DSL. One of these gadget attributes is the **Name** attribute, which should unambiguously identify the gadget for your application resources.

To ensure proper identification, the **Name** attribute supports the use of a **gadget path** for **Group** gadgets and **Menu** gadgets. A gadget path names both the gadget and the gadgets that must be traversed to reach it. This is similar to a Unix path, which names both the file and the directories that must be traversed to reach the file.

The purpose of the gadget path is to pass gadget values to a callback method. A callback is an operation performed by the desktop when an operation is requested from a dialog gadget. The callback invokes the appropriate method, which then performs the requested operation. The callback method uses the gadget path to manipulate the invoking gadget.

A gadget path is the list of the gadget names in the hierarchy that contain the desired gadget. The path begins with the outermost enclosing gadget. The path then contains the next enclosing gadget, and so on. The last named gadget is the target gadget.

Separate each gadget name with a period (.). For example, if you create a gadget called **user** that contains a gadget called **password** that in turn contains a third gadget called **max**, the gadget path to the **max** gadget is **user.password.max**.

Gadget paths must be unique within a single instance of a dialog. It is not necessary, however, for gadget paths to be unique among different dialogs or multiple instances of the same dialog. Each gadget is identified by two parameters: the gadget path and the dialog name. Consider the fragment below.

```
1 Command Dialog {
2   Attributes {
3     Name = main;
4     Title = "Hi there";
5   }
6 }
7 Menu {
8   Attributes {
9     Name = m;
```
Logical Organization of a Dialog Specification

10    TearOff = YES;
11    Title = "A Menu";
12   }
13   Gadgets {
14    Button {
15     Name = b;
16     Title = "Push Me";
17    }
18    Menu {
19     Attributes {
20      Name = m2;
21      Title = "A Submenu";
22      TearOff = YES;
23     }
24     Gadgets {
25      Button {
26       Name = b2;
27       Title = "Don't push me";
28      }
29     }
30    }
31   }
32 }
33 }

In this example, the gadget hierarchy begins with the dialog name **main**. The **main** dialog contains a menu named **m**, which contains a button gadget named **b**. Thus, the button gadget has the following gadget path: **main.m.b**.

The example code fragment contains several other gadgets nested within the main dialog gadget hierarchy. Each of these gadgets has a unique path, which describes its position in the hierarchy as shown in the table below.

<table>
<thead>
<tr>
<th>GUI Component</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Dialog (<strong>main</strong>)</td>
<td><strong>main</strong></td>
</tr>
<tr>
<td>First Dialog Menu (<strong>m</strong>)</td>
<td><strong>main.m</strong></td>
</tr>
<tr>
<td>First Menu Button (<strong>b</strong>)</td>
<td><strong>main.m.b</strong></td>
</tr>
</tbody>
</table>
References to gadgets in each dialog instance are therefore not ambiguous; specifying a dialog name and a gadget path refers unambiguously to a single gadget.

## Lexical Components of DSL

DSL supports the following lexical components:

- Blocks
- Comments
- Special characters
- Identifiers
- Strings
- Desktop commands

### Blocks

The basic structure of the DSL language is the **block**. A block is a set of statements enclosed in curly braces ({ }). Each statement in the block is terminated by a semicolon (;). The type of block is dictated by the keyword that precedes the block. For example, a block of variables begins with the keyword **Variables**.

```plaintext
Variables {
    CString PAGE_RESULTS = get_results@();
    CString REGION_LABEL = get_label@();
}
```

Blocks can occur many times in any order. For example, twenty **Variables** blocks, each declaring a single variable, has the same effect as one **Variables** block that declares twenty variables.
Lexical Components of DSL

Comments

DSL ignores all comments. The two-character sequence /* begins a comment that continues until terminated by the two-character sequence */. This type of comment can span multiple lines. Do not nest comments within this comment style.

Special Characters

The following table shows the special characters that are reserved by the DSL compiler.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Separates components of gadget path names.</td>
</tr>
<tr>
<td>$</td>
<td>Indicates that the following symbol is a reference to a variable.</td>
</tr>
<tr>
<td>;</td>
<td>Terminates a statement (for example, a single variable definition or a single attribute definition).</td>
</tr>
<tr>
<td>&lt;</td>
<td>Indicates that a callback method expects data on standard input.</td>
</tr>
<tr>
<td>(</td>
<td>Begins the arguments to a callback.</td>
</tr>
<tr>
<td>)</td>
<td>Terminates the arguments to a callback.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Delimits a string.</td>
</tr>
<tr>
<td>@</td>
<td>Follows the method name of a legacy method callback.</td>
</tr>
<tr>
<td>{</td>
<td>Begins certain syntactic “blocks” or “groupings” in DSL, such as the dialog body, a gadget’s body, a list of variable definitions, a list of attribute definitions, or an item’s contents. Also brackets the optional value for a Choices attribute.</td>
</tr>
<tr>
<td>}</td>
<td>Terminates syntactic blocks started by a left brace.</td>
</tr>
<tr>
<td>,</td>
<td>Separates a list of components, such as items, reasons, callbacks, expressions, or choices.</td>
</tr>
<tr>
<td>=</td>
<td>Associates a name (on the left) with a value (on the right) in variable and attribute definitions.</td>
</tr>
</tbody>
</table>
Dialog Callbacks

Identifiers

An identifier (also called a symbol) is a sequence of at least one character that does not contain any keywords, white space or special characters. The only non-alphanumeric character allowed in identifiers is the underscore character.

An identifier prefixed by a dollar sign ($) is a variable. The following variables are reserved and cannot be redefined:

- $drops
- $selects
- $icon
- $owner

These variables (also known as special variables) are part of a larger set of variables known as desktop variables.

Strings

A DSL string is any sequence of characters enclosed between pairs of double-quote characters ("""). A string can span several lines; actual newline characters in strings are ignored. There are several special sequences of characters you can use in strings:

- Two back-slash characters (\) yield a single back-slash character in the string.
- A back-slash followed by a lower-case letter n (\n) yields a single newline character in the string.
- A single back-slash character followed by a double-quote character (\") yields a double-quote character in the string and does not terminate the string.

Dialog Callbacks

When users request an operation from a gadget, the TME 10 desktop performs an operation known as a callback. A callback is the means by which dialogs and gadgets respond to user input. Callbacks typically transfer data and control out of the user interface and back to underlying application objects. Callbacks specify an
operation, which can take the form of an internal command or an external program.

Internal commands are also called *desktop commands*. Desktop commands affect only the user interface; they do not communicate with the application. External programs consist of methods invoked on application resources. For example, when a user presses a button to create a user, the dialog invokes an external callback to create an instance of the user resource type.

Callback specifications are stored as dialog and gadget attributes. The TME 10 desktop supports two types of callbacks:

- **Method callbacks** — Callbacks to methods outside the TME 10 desktop.
- **Desktop command callbacks** — Callbacks to commands within the TME 10 desktop.

Callbacks are usually defined through the **Commands** attribute of a gadget.

```plaintext
Menu {
  Name = my_menu;
  Button {
    Name = b1;
    Title = "Show Properties";
    Commands = show_properties@(),
    another_method(),
    dtc_dismiss($owner, my_menu, $self);
  }
}
```

In the example DSL above, TME 10 invokes three callback methods when the **Show Properties** button is pressed (**show_properties**, **another_method**, and **dtc_dismiss**).

### Method Callbacks

There are several types of callback available from DSL. However, the only type of callback that you can use with TME 10 AEF is the **legacy callback**. A legacy callback is a program that takes arguments on the command line, can read from standard input, and can write to standard
output. To use other types of callback methods, you must use TME 10 ADE.

The syntax for a legacy callback is shown below.

```
method @ ( args ) [< input ]
```

The `method` is the method name. The method name must always be followed by an `@` character. The method is invoked in the context of the object that posted the dialog. For example, if you are viewing a dialog on the `moab` managed node, the dialog’s callback methods are invoked on the `moab` resource instance.

The `args` parameter is an optional list of comma-separated arguments to the method, enclosed in parentheses (you must always include the parentheses, even if they are empty).

If the method is to receive input to its standard input, you can supply it using the `input` parameter. You must precede the input with a angle bracket (`<`).

When successful, the method should return an exit code of zero. Otherwise, it should return an exit code of one.

See Chapter 7, “Custom Callback Methods” for information on how to create and install custom callback methods.

**Desktop Command Callbacks**

Desktop commands are passed directly from your dialog to the desktop as callbacks. They are very different from callbacks to methods, which are external programs invoked by the TME 10 desktop. Desktop command callbacks provide better performance and are less complicated than callbacks to external methods. Using them whenever possible therefore provides the obvious advantage of better performance and greater reliability from your desktop.

The syntax of a desktop command callback is shown below.

```
command-name ( [params] )
```

The `command-name` is the name of the callback, and `params` is an optional list of comma-separated parameters enclosed in parentheses. The parentheses are required even if you do not supply any parameters.
Dialog Callbacks

The following DSL code fragment depicts a typical desktop command callback for a **CommandButton** gadget:

```plaintext
1  CommandButton {
2    Name = dismiss;
3    Title = "Dismiss";
4    Commands = dtc_dismiss($owner,
5                       check_policy_results,
6                       $self);
7  }
```

The **Commands** attribute on line 4 specifies the callbacks TME 10 invokes when the button is pressed. In this case, TME 10 invokes the **dtc_dismiss** command to dismiss (close) the dialog.

The following table lists the available desktop commands.

<table>
<thead>
<tr>
<th>Desktop Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtc_busy</td>
<td>Makes a gadget busy</td>
</tr>
<tr>
<td>dtc_change</td>
<td>Changes attributes of a posted dialog</td>
</tr>
<tr>
<td>dtc_clear</td>
<td>Removes icons from a collection</td>
</tr>
<tr>
<td>dtc_clipboard</td>
<td>Performs cut, paste, and copy operations on a collection</td>
</tr>
<tr>
<td>dtc_delete</td>
<td>Deletes a gadget from a posted dialog</td>
</tr>
<tr>
<td>dtc_deselect</td>
<td>Deselects icons in a specified collection</td>
</tr>
<tr>
<td>dtc_dismiss</td>
<td>Dismisses a dialog; that is, it returns the dialog to a defined yet invisible state</td>
</tr>
<tr>
<td>dtc_dismiss_all</td>
<td>Removes all dialogs from the screen</td>
</tr>
<tr>
<td>dtc_exit</td>
<td>Quietly causes the TME 10 desktop to exit</td>
</tr>
<tr>
<td>dtc_free</td>
<td>Frees a dialog; that is it returns the dialog to an undefined state</td>
</tr>
<tr>
<td>dtc_move_gadget</td>
<td>Moves a gadget from one location to another within a dialog</td>
</tr>
</tbody>
</table>
### Gadget Library Callbacks

Gadget library callbacks are callbacks that are made directly to your application much like desktop command callbacks. For reasons that are beyond the scope of this document, support for desktop command callbacks will eventually be eliminated, and so gadget library callbacks are recommended for all new AEF customizations.

Gadget library callbacks are identified with an ampersand sign (&), as shown in the following code fragment:

```plaintext
Menu {
  Name = my_menu;
  Button {
    Name = b1;
    Title = "Show Properties";
  }
}
```

<table>
<thead>
<tr>
<th>Desktop Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtc_nop</td>
<td>No operation; enables you to provide diagnostics for the current dialog using the parameters in the command signature</td>
</tr>
<tr>
<td>dtc_play</td>
<td>Plays a sound file</td>
</tr>
<tr>
<td>dtc_present</td>
<td>Presents (displays) a defined, invisible dialog</td>
</tr>
<tr>
<td>dtc_present_all</td>
<td>Presents all dialogs dismissed using dtc_dismiss_all</td>
</tr>
<tr>
<td>dtc_quit</td>
<td>Presents confirmation dialog about quitting</td>
</tr>
<tr>
<td>dtc_refresh</td>
<td>Instructs the TME 10 desktop to invoke the refresh callbacks for this dialog</td>
</tr>
<tr>
<td>dtc_return</td>
<td>Dismisses the current dialog and presents its parent dialog</td>
</tr>
<tr>
<td>dtc_select</td>
<td>Selects one or more gadgets in a specified collection</td>
</tr>
<tr>
<td>dtc_table_select_axis</td>
<td>Selects a table row or column</td>
</tr>
<tr>
<td>dtc_unbusy</td>
<td>Causes a gadget to become unbusy</td>
</tr>
</tbody>
</table>
Some gadget library callbacks can be used only in a specific gadget such as a Button or only in a dialog. The following sections present each gadget library callback supported by TME 10 AEF and the gadgets in which they can be used.

**Callbacks for All Gadgets**

- **create&(name, descriptor, parameters)**
  
  This callback loads a dialog descriptor, assigns it a name, binds all dialog variables given in parameters, makes the newly loaded dialog a child of the current dialog, and then presents it.

- **load&(name, descriptor, parameters)**
  
  This callback loads a dialog descriptor, assigns it a name, binds all dialog variables given in parameters, and makes the newly loaded dialog a child of the current dialog. The dialog can then be presented using the present built-in callback (described in “Dialog Callbacks” on page 4-30).

- **nop&(...)**
  
  This callback invokes the dtc_nop desktop command. See “Desktop Command Callbacks” on page 4-23 for a summary of dtc_nop and other desktop commands.

- **set_background&(background_string);**
  
  This callback sets the Background attribute for the specified gadget to background_string.

- **set_foreground&(foreground_string);**
  
  This callback sets the Foreground attribute for the specified gadget to foreground-string.

- **set_bitmap_title&(bitmap_title);**
  
  This callback sets the BitmapTitle attribute for the specified gadget to bitmap_title.
- `set_help_message&(help-string);`
  This callback sets the **HelpMessage** attribute for the given gadget to the literal `help_string`.

- `set_help_message&(catalog, default_text, key, parameters);`
  This callback sets the **HelpMessage** attribute for the specified gadget to the internationalized string given by `catalog`, `default_text`, and `key`, with arguments to the message given in `parameters`.

- `set_title&(title_string)`
  This callback sets the **Title** attribute for the given gadget to the `title_string`.

- `set_title&(catalog, default_text, key, parameters);`
  This callback sets the **Title** attribute for the given gadget to the internationalized string given by `catalog`, `default_text`, and `key`, with arguments to the message given in `parameters`.

- `set_border&(YES|NO);`
  This callback sets the **Border** attribute for the specified gadget to the given value.

- `set_sensitive&(YES|NO);`
  This callback sets the **Sensitive** attribute for the specified gadget to the specified value.

- `set_title_pos&(LEFT|CENTER|TOP|RIGHT|BOTTOM|STRETCH);`
  This callback sets the **TitlePos** attribute for the specified gadget to the specified value.

- `set_visible&(YES|NO);`
  This callback sets the **Visible** attribute for the specified gadget to the specified value.
Dialog Callbacks

**Button Gadget Callbacks**

- `add_command&(dialog-path, gadget-path, callback-name, parameters);`
  
  This callback creates a callback and adds it to the Button Commands attribute.

**Choice and List Gadget Callbacks**

- `add_choices&(dialog_path, chooser_path);`
  
  This callback adds the choices currently selected in `dialog_path` and `chooser_path` to this Choice or List gadget.

- `remove_choices&();`
  
  This callback removes the current selections from this choice list.

**Group Gadget Callbacks**

- `set_layout&(HORIZONTAL|VERTICAL|PACKED|GRID);`
  
  This callback sets the layout for this Group attribute to the given value.

- `set_alignment&(LEFT|CENTER|TOP|RIGHT|BOTTOM|STRETCH);`
  
  This callback sets the Alignment attribute for the current Group to the given value.

- `set_scrolling&(YES|NO);`
  
  This callback sets the Scrolling attribute for the current Group to the given value.

- `set_grid_height&(grid-height);`
  
  This callback sets the GridHeight attribute for the current Group to the specified value.

- `set_grid_width&(grid-width);`
  
  This callback sets the GridWidth attribute for the current Group to the specified value.
Dialog Callbacks

- delete_gadget&(gadget-name);
  This callback deletes the given gadget in this Group.

Switch Gadget Callbacks

- set_value&(on_or_off_string)
  This callback sets the Value attribute for this Switch to on_or_off_string, which must be either the value of the On attribute or the value of the Off attribute.

- set_on&(on_string)
  This callback sets the value of the On attribute for the current Switch to on_string.

- set_off&(off_string)
  This callback sets the value of the Off attribute for the current Switch to off_string.

Table Gadget Callbacks

- set_column_selection&(SINGLE_SELECT|MULTIPLE_SELECT|EXTENDED_SELECT)
  This callback sets the ColumnSelection attribute for the current Table to the specified value.

- set_column_width&(width)
  This callback sets the ColumnWidth attribute for this Table to the specified value.

- set_columns&(num-columns)
  This callback sets the Columns attribute for the current Table to the specified value.

- set_layout_policy&(COLUMN_MAJOR, ROW_MAJOR)
  This callback sets the LayoutPolicy attribute for the current Table to the given value.

- set_left_column&(TableColumn-name)
  This callback sets the LeftColumn attribute for the current Table to the specified value.
Dialog Callbacks

- set_row_selection&(SINGLE_SELECT|MULTIPLE_SELECT|EXTENDED_SELECT)
  This callback sets the RowSelection attribute for the current Table to the specified value.

- set_rows&(num_rows)
  This callback sets the Rows attribute for current Table to the specified value.

- set_selection_mode&(SELECT_NONE,SELECT_ROW, SELECT_COLUMN,SELECT_BOTH)
  This callback sets the SelectionMode attribute for the current Table to the specified value.

- set_top_row&(TableRow_name)
  This callback sets the TopRow attribute for the current Table to the given TableRow_name.

- set_glyph&(glyph_name);
  This callback sets the Glyph attribute for this TableRow or TableColumn to glyph_name

- set_alignment&(LEFT|CENTER|RIGHT);
  This callback sets the Alignment attribute for current TableRow or TableColumn to the specified value.

- set_scrolling&(YES|NO);
  This callback sets the Scrolling attribute for the current TableRow or TableColumn to the specified value.

- set_read_only&(YES|NO);
  This callback sets the ReadOnly attribute for the current TableRow or TableColumn to the given value.

Dialog Callbacks

- present&();
  This callback presents the current dialog. The dialog must have been previously loaded with a load callback (glBase).
Managing Callback Errors

A legacy method should use traditional error code semantics; it should return an exit code of zero if no error is detected, or non-zero if an error is detected.

If the method returns a non-zero error code the TME 10 desktop intercepts the exit code and displays a simple error dialog that contains a non-specific error message. In this case, the TME 10 desktop basically manages the error for the application.

The TME 10 error dialog is a modal dialog; once displayed the user must dismiss it from the GUI. The offending dialog will remain, however, after the TME 10 error dialog is dismissed.

Alternatively, the application can be responsible for managing the error by posting dialogs that enable users to recover. After addressing the error using the dialogs presented by the application, the user is returned to the offending dialog. The user then has the option of attempting to retry the action aborting it.

Chapter 7, “Custom Callback Methods” discusses techniques and commands that will help you perform error handling within custom callback methods.

Bitmap Support in DSL

Applications frequently employ bitmaps in their dialogs, since they are an efficient means of presenting information graphically.
Color Support in DSL

Bitmaps are often used as the title of a dialog or gadget. For example, many dialogs use button bitmaps on gadgets that transfer items to and from a list.

To use a bitmap as the title of a gadget, set the gadget’s `BitmapTitle` attribute using the `BITMAP_FILE` directive.

```plaintext
BitmapTitle = BITMAP_FILE(file_name);
```

The `file_name` parameter is the name of the bitmap file, which must be in `X pixmap format` (XPM).

Many resources have bitmaps installed for use by the resource’s dialogs. To use a bitmap that is installed on a resource, use the following syntax:

```plaintext
BitmapTitle = bitmap_name;
```

The `bitmap_name` parameter is the internal name of the bitmap.

---

Color Support in DSL

DSL supports color specification for dialogs and gadgets. The color can be specified using two attributes: `Background` and `Foreground`. `Background` controls the color of the dialog or gadget, and `Foreground` controls the color of anything drawn in the dialog or gadget.

TME 10 supports the RGB color model, so you can specify color values in two ways:

- Specify color values in hexadecimal:
  ```plaintext
  Foreground = #00ff00
  ```
The value specification always takes the form #rrggbb. The RGB color model specifies three components for each color: red, green, and blue. The rr variable specifies the red component, the gg specifies the green component, and bb specifies the blue component.

The hexadecimal values specify the component intensity: 00 represents off and ff represents full on. Thus the example (Foreground = #00ff00) specifies a green foreground, because the green component is turned full on and the red and blue components are turned off.

All shades of all colors can be produced by changing the intensity of one or more color components. White is produced by turning all three components full on, and black is produced by turning all three components off.

Other colors are produced by specifying some value between 00 and ff for each component. Light blue, for example, is produced by the following hexadecimal specification:

Foreground = #ADD8E6

Each color component in this example is set to an intensity such that when they are combined, they produce light blue. The blue component has the greatest intensity, and thus the resulting color is perceived as a shade of blue.

Mixing the color components to obtain a specific color shade using hexadecimal values can be complex. You should therefore use this technique only when the required color is not available using the second technique. See your X users guide for more details about color specification and the RGB color model.

Specify a color name:

Foreground = name

DSL supports the following colors by name:

<table>
<thead>
<tr>
<th>Color</th>
<th>Color</th>
<th>Color</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>aliceblue</td>
<td>antquwhite</td>
<td>antquwhite1</td>
<td>antquwhite2</td>
</tr>
<tr>
<td>antquwhite3</td>
<td>antquwhite4</td>
<td>aquamarine</td>
<td>aquamarine1</td>
</tr>
</tbody>
</table>
## Color Support in DSL

<table>
<thead>
<tr>
<th>aquamarine2</th>
<th>aquamarine3</th>
<th>aquamarine4</th>
<th>azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>azure1</td>
<td>azure2</td>
<td>azure3</td>
<td>azure4</td>
</tr>
<tr>
<td>beige</td>
<td>bisque</td>
<td>bisque1</td>
<td>bisque2</td>
</tr>
<tr>
<td>bisque3</td>
<td>bisque4</td>
<td>black</td>
<td>blanchedalmond</td>
</tr>
<tr>
<td>blue</td>
<td>blue1</td>
<td>blue2</td>
<td>blue3</td>
</tr>
<tr>
<td>blue4</td>
<td>blueviolet</td>
<td>brown</td>
<td>brown1</td>
</tr>
<tr>
<td>brown2</td>
<td>brown3</td>
<td>brown4</td>
<td>burlywood</td>
</tr>
<tr>
<td>burlywood1</td>
<td>burlywood2</td>
<td>burlywood3</td>
<td>burlywood4</td>
</tr>
<tr>
<td>cadetblue</td>
<td>cadetblue1</td>
<td>cadetblue2</td>
<td>cadetblue3</td>
</tr>
<tr>
<td>cadetblue4</td>
<td>chartreuse</td>
<td>chartreuse1</td>
<td>chartreuse2</td>
</tr>
<tr>
<td>chartreuse3</td>
<td>chartreuse4</td>
<td>chocolate</td>
<td>chocolate1</td>
</tr>
<tr>
<td>chocolate2</td>
<td>chocolate3</td>
<td>chocolate4</td>
<td>coral</td>
</tr>
<tr>
<td>coral1</td>
<td>coral2</td>
<td>coral3</td>
<td>coral4</td>
</tr>
<tr>
<td>cornflowerblue</td>
<td>cornsilk</td>
<td>cornsilk1</td>
<td>cornsilk2</td>
</tr>
<tr>
<td>cornsilk3</td>
<td>cornsilk4</td>
<td>cyan</td>
<td>cyan1</td>
</tr>
<tr>
<td>cyan2</td>
<td>cyan3</td>
<td>cyan4</td>
<td>darkgoldenrod</td>
</tr>
<tr>
<td>darkgoldenrod1</td>
<td>darkgoldenrod2</td>
<td>darkgoldenrod3</td>
<td>darkgoldenrod4</td>
</tr>
<tr>
<td>darkgreen</td>
<td>darkkhaki</td>
<td>darkolivegreen</td>
<td>darkolivegreen1</td>
</tr>
<tr>
<td>darkolivegreen2</td>
<td>darkolivegreen3</td>
<td>darkolivegreen4</td>
<td>darkorange</td>
</tr>
<tr>
<td>darkorange1</td>
<td>darkorange2</td>
<td>darkorange3</td>
<td>darkorange4</td>
</tr>
<tr>
<td>darkorchid</td>
<td>darkorchid1</td>
<td>darkorchid2</td>
<td>darkorchid3</td>
</tr>
<tr>
<td>darkorchid4</td>
<td>darksalmon</td>
<td>darkseagreen</td>
<td>darkseagreen1</td>
</tr>
<tr>
<td>darkseagreen2</td>
<td>darkseagreen3</td>
<td>darkseagreen4</td>
<td>darkslateblue</td>
</tr>
<tr>
<td>darkslategray</td>
<td>darkslategray1</td>
<td>darkslategray2</td>
<td>darkslategray3</td>
</tr>
<tr>
<td>darkslategray4</td>
<td>darkslategrey</td>
<td>darkturquoise</td>
<td>darkviolet</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>Color</td>
<td>Color</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>deeppink</td>
<td>deeppink1</td>
<td>deeppink2</td>
<td>deeppink3</td>
</tr>
<tr>
<td>deeppink4</td>
<td>deepskyblue</td>
<td>deepskyblue1</td>
<td>deepskyblue2</td>
</tr>
<tr>
<td>deepskyblue3</td>
<td>deepskyblue4</td>
<td>dimgray</td>
<td>dimgrey</td>
</tr>
<tr>
<td>dodgerblue</td>
<td>dodgerblue1</td>
<td>dodgerblue2</td>
<td>dodgerblue3</td>
</tr>
<tr>
<td>dodgerblue4</td>
<td>firebrick</td>
<td>firebrick1</td>
<td>firebrick2</td>
</tr>
<tr>
<td>firebrick3</td>
<td>firebrick4</td>
<td>floralwhite</td>
<td>forestgreen</td>
</tr>
<tr>
<td>gainsboro</td>
<td>ghostwhite</td>
<td>gold</td>
<td>gold1</td>
</tr>
<tr>
<td>gold2</td>
<td>gold3</td>
<td>gold4</td>
<td>goldenrod</td>
</tr>
<tr>
<td>goldenrod1</td>
<td>goldenrod2</td>
<td>goldenrod3</td>
<td>goldenrod4</td>
</tr>
<tr>
<td>gray</td>
<td>gray0</td>
<td>gray1</td>
<td>gray10</td>
</tr>
<tr>
<td>gray100</td>
<td>gray11</td>
<td>gray12</td>
<td>gray13</td>
</tr>
<tr>
<td>gray14</td>
<td>gray15</td>
<td>gray16</td>
<td>gray17</td>
</tr>
<tr>
<td>gray18</td>
<td>gray19</td>
<td>gray2</td>
<td>gray20</td>
</tr>
<tr>
<td>gray21</td>
<td>gray22</td>
<td>gray23</td>
<td>gray24</td>
</tr>
<tr>
<td>gray25</td>
<td>gray26</td>
<td>gray27</td>
<td>gray28</td>
</tr>
<tr>
<td>gray29</td>
<td>gray3</td>
<td>gray30</td>
<td>gray31</td>
</tr>
<tr>
<td>gray32</td>
<td>gray33</td>
<td>gray34</td>
<td>gray35</td>
</tr>
<tr>
<td>gray36</td>
<td>gray37</td>
<td>gray38</td>
<td>gray39</td>
</tr>
<tr>
<td>gray4</td>
<td>gray40</td>
<td>gray41</td>
<td>gray42</td>
</tr>
<tr>
<td>gray43</td>
<td>gray44</td>
<td>gray45</td>
<td>gray46</td>
</tr>
<tr>
<td>gray47</td>
<td>gray48</td>
<td>gray49</td>
<td>gray5</td>
</tr>
<tr>
<td>gray50</td>
<td>gray51</td>
<td>gray52</td>
<td>gray53</td>
</tr>
<tr>
<td>gray54</td>
<td>gray55</td>
<td>gray56</td>
<td>gray57</td>
</tr>
<tr>
<td>gray58</td>
<td>gray59</td>
<td>gray6</td>
<td>gray60</td>
</tr>
<tr>
<td>gray61</td>
<td>gray62</td>
<td>gray63</td>
<td>gray64</td>
</tr>
</tbody>
</table>
Color Support in DSL

<table>
<thead>
<tr>
<th>gray65</th>
<th>gray66</th>
<th>gray67</th>
<th>gray68</th>
</tr>
</thead>
<tbody>
<tr>
<td>gray69</td>
<td>gray7</td>
<td>gray70</td>
<td>gray71</td>
</tr>
<tr>
<td>gray72</td>
<td>gray73</td>
<td>gray74</td>
<td>gray75</td>
</tr>
<tr>
<td>gray76</td>
<td>gray77</td>
<td>gray78</td>
<td>gray79</td>
</tr>
<tr>
<td>gray8</td>
<td>gray80</td>
<td>gray81</td>
<td>gray82</td>
</tr>
<tr>
<td>gray83</td>
<td>gray84</td>
<td>gray85</td>
<td>gray86</td>
</tr>
<tr>
<td>gray87</td>
<td>gray88</td>
<td>gray89</td>
<td>gray9</td>
</tr>
<tr>
<td>gray90</td>
<td>gray91</td>
<td>gray92</td>
<td>gray93</td>
</tr>
<tr>
<td>gray94</td>
<td>gray95</td>
<td>gray96</td>
<td>gray97</td>
</tr>
<tr>
<td>gray98</td>
<td>gray99</td>
<td>green</td>
<td>green1</td>
</tr>
<tr>
<td>grey2</td>
<td>green3</td>
<td>green4</td>
<td>greenyellow</td>
</tr>
<tr>
<td>grey</td>
<td>grey0</td>
<td>grey1</td>
<td>grey10</td>
</tr>
<tr>
<td>grey100</td>
<td>grey11</td>
<td>grey12</td>
<td>grey13</td>
</tr>
<tr>
<td>grey14</td>
<td>grey15</td>
<td>grey16</td>
<td>grey17</td>
</tr>
<tr>
<td>grey18</td>
<td>grey19</td>
<td>grey2</td>
<td>grey20</td>
</tr>
<tr>
<td>grey21</td>
<td>grey22</td>
<td>grey23</td>
<td>grey24</td>
</tr>
<tr>
<td>grey25</td>
<td>grey26</td>
<td>grey27</td>
<td>grey28</td>
</tr>
<tr>
<td>grey29</td>
<td>grey3</td>
<td>grey30</td>
<td>grey31</td>
</tr>
<tr>
<td>grey32</td>
<td>grey33</td>
<td>grey34</td>
<td>grey35</td>
</tr>
<tr>
<td>grey36</td>
<td>grey37</td>
<td>grey38</td>
<td>grey39</td>
</tr>
<tr>
<td>grey4</td>
<td>grey40</td>
<td>grey41</td>
<td>grey42</td>
</tr>
<tr>
<td>grey43</td>
<td>grey44</td>
<td>grey45</td>
<td>grey46</td>
</tr>
<tr>
<td>grey47</td>
<td>grey48</td>
<td>grey49</td>
<td>grey5</td>
</tr>
<tr>
<td>grey50</td>
<td>grey51</td>
<td>grey52</td>
<td>grey53</td>
</tr>
<tr>
<td>grey54</td>
<td>grey55</td>
<td>grey56</td>
<td>grey57</td>
</tr>
</tbody>
</table>
### Color Support in DSL

<table>
<thead>
<tr>
<th>grey58</th>
<th>grey59</th>
<th>grey6</th>
<th>grey60</th>
</tr>
</thead>
<tbody>
<tr>
<td>grey61</td>
<td>grey62</td>
<td>grey63</td>
<td>grey64</td>
</tr>
<tr>
<td>grey65</td>
<td>grey66</td>
<td>grey67</td>
<td>grey68</td>
</tr>
<tr>
<td>grey69</td>
<td>grey7</td>
<td>grey70</td>
<td>grey71</td>
</tr>
<tr>
<td>grey72</td>
<td>grey73</td>
<td>grey74</td>
<td>grey75</td>
</tr>
<tr>
<td>grey76</td>
<td>grey77</td>
<td>grey78</td>
<td>grey79</td>
</tr>
<tr>
<td>grey8</td>
<td>grey80</td>
<td>grey81</td>
<td>grey82</td>
</tr>
<tr>
<td>grey83</td>
<td>grey84</td>
<td>grey85</td>
<td>grey86</td>
</tr>
<tr>
<td>grey87</td>
<td>grey88</td>
<td>grey89</td>
<td>grey9</td>
</tr>
<tr>
<td>grey90</td>
<td>grey91</td>
<td>grey92</td>
<td>grey93</td>
</tr>
<tr>
<td>grey94</td>
<td>grey95</td>
<td>grey96</td>
<td>grey97</td>
</tr>
<tr>
<td>grey98</td>
<td>grey99</td>
<td>honeydew</td>
<td>honeydew1</td>
</tr>
<tr>
<td>honeydew2</td>
<td>honeydew3</td>
<td>honeydew4</td>
<td>hotpink</td>
</tr>
<tr>
<td>hotpink1</td>
<td>hotpink2</td>
<td>hotpink3</td>
<td>hotpink4</td>
</tr>
<tr>
<td>indianred</td>
<td>indianred1</td>
<td>indianred2</td>
<td>indianred3</td>
</tr>
<tr>
<td>indianred4</td>
<td>ivory</td>
<td>ivory1</td>
<td>ivory2</td>
</tr>
<tr>
<td>ivory3</td>
<td>ivory4</td>
<td>khaki</td>
<td>khaki1</td>
</tr>
<tr>
<td>khaki2</td>
<td>khaki3</td>
<td>khaki4</td>
<td>lavender</td>
</tr>
<tr>
<td>lavenderblush</td>
<td>lavenderblush1</td>
<td>lavenderblush2</td>
<td>lavenderblush3</td>
</tr>
<tr>
<td>lavenderblush4</td>
<td>lawngreen</td>
<td>lemonchiffon</td>
<td>lemonchiffon1</td>
</tr>
<tr>
<td>lemonchiffon2</td>
<td>lemonchiffon3</td>
<td>lemonchiffon4</td>
<td>lightblue</td>
</tr>
<tr>
<td>lightblue1</td>
<td>lightblue2</td>
<td>lightblue3</td>
<td>lightblue4</td>
</tr>
<tr>
<td>lightcoral</td>
<td>lightcyan</td>
<td>lightcyan1</td>
<td>lightcyan2</td>
</tr>
<tr>
<td>lightcyan3</td>
<td>lightcyan4</td>
<td>lightgoldenrod</td>
<td>lightgoldenrod1</td>
</tr>
<tr>
<td>lightgoldenrod2</td>
<td>lightgoldenrod3</td>
<td>lightgoldenrod4</td>
<td>lightgoldenrodyellow</td>
</tr>
</tbody>
</table>
## Color Support in DSL

<table>
<thead>
<tr>
<th>lightgray</th>
<th>lightgrey</th>
<th>lightpink</th>
<th>lightpink1</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightpink2</td>
<td>lightpink3</td>
<td>lightpink4</td>
<td>lightsalmon</td>
</tr>
<tr>
<td>lightsalmon1</td>
<td>lightsalmon2</td>
<td>lightsalmon3</td>
<td>lightsalmon4</td>
</tr>
<tr>
<td>lightseagreen</td>
<td>lightskyblue</td>
<td>lightskyblue1</td>
<td>lightskyblue2</td>
</tr>
<tr>
<td>lightskyblue3</td>
<td>lightskyblue4</td>
<td>lightslateblue</td>
<td>lightslategray</td>
</tr>
<tr>
<td>lightslategrey</td>
<td>lightsteelblue</td>
<td>lightsteelblue1</td>
<td>lightsteelblue2</td>
</tr>
<tr>
<td>lightsteelblue3</td>
<td>lightsteelblue4</td>
<td>lightyellow</td>
<td>lightyellow1</td>
</tr>
<tr>
<td>lightyellow2</td>
<td>lightyellow3</td>
<td>lightyellow4</td>
<td>limegreen</td>
</tr>
<tr>
<td>linen</td>
<td>magenta</td>
<td>magenta1</td>
<td>magenta2</td>
</tr>
<tr>
<td>magenta3</td>
<td>magenta4</td>
<td>maroon</td>
<td>maroon1</td>
</tr>
<tr>
<td>maroon2</td>
<td>maroon3</td>
<td>maroon4</td>
<td>mediumaquamarine</td>
</tr>
<tr>
<td>mediumblue</td>
<td>mediumorchid</td>
<td>mediumorchid1</td>
<td>mediumorchid2</td>
</tr>
<tr>
<td>mediumorchid3</td>
<td>mediumorchid4</td>
<td>mediumpurple</td>
<td>mediumpurple1</td>
</tr>
<tr>
<td>mediumpurple2</td>
<td>mediumpurple3</td>
<td>mediumpurple4</td>
<td>mediumseagreen</td>
</tr>
<tr>
<td>mediumslateblue</td>
<td>mediumspringgreen</td>
<td>mediumturquoise</td>
<td>mediumvioletred</td>
</tr>
<tr>
<td>midnightblue</td>
<td>mintcream</td>
<td>mistyrose</td>
<td>mistyrose1</td>
</tr>
<tr>
<td>mistyrose2</td>
<td>mistyrose3</td>
<td>mistyrose4</td>
<td>moccasin</td>
</tr>
<tr>
<td>navajowhite</td>
<td>navajowhite1</td>
<td>navajowhite2</td>
<td>navajowhite3</td>
</tr>
<tr>
<td>navajowhite4</td>
<td>navy</td>
<td>navyblue</td>
<td>oldlace</td>
</tr>
<tr>
<td>olivedrab</td>
<td>olivedrab1</td>
<td>olivedrab2</td>
<td>olivedrab3</td>
</tr>
<tr>
<td>olivedrab4</td>
<td>orange</td>
<td>orange1</td>
<td>orange2</td>
</tr>
<tr>
<td>orange3</td>
<td>orange4</td>
<td>orangered</td>
<td>orangered1</td>
</tr>
<tr>
<td>orangered2</td>
<td>orangered3</td>
<td>orangered4</td>
<td>orchid</td>
</tr>
<tr>
<td>orchid1</td>
<td>orchid2</td>
<td>orchid3</td>
<td>orchid4</td>
</tr>
<tr>
<td>palegoldenrod</td>
<td>palegreen</td>
<td>palegreen1</td>
<td>palegreen2</td>
</tr>
<tr>
<td>palegreen3</td>
<td>palegreen4</td>
<td>paleturquoise</td>
<td>paleturquoise1</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>paleturquoise2</td>
<td>paleturquoise3</td>
<td>paleturquoise4</td>
<td>palevioletred</td>
</tr>
<tr>
<td>palevioletred1</td>
<td>palevioletred2</td>
<td>palevioletred3</td>
<td>palevioletred4</td>
</tr>
<tr>
<td>papayawhip</td>
<td>peachpuff</td>
<td>peachpuff1</td>
<td>peachpuff2</td>
</tr>
<tr>
<td>peachpuff3</td>
<td>peachpuff4</td>
<td>peru</td>
<td>pink</td>
</tr>
<tr>
<td>pink1</td>
<td>pink2</td>
<td>pink3</td>
<td>pink4</td>
</tr>
<tr>
<td>plum</td>
<td>plum1</td>
<td>plum2</td>
<td>plum3</td>
</tr>
<tr>
<td>plum4</td>
<td>powderblue</td>
<td>purple</td>
<td>purple1</td>
</tr>
<tr>
<td>purple2</td>
<td>purple3</td>
<td>purple4</td>
<td>red</td>
</tr>
<tr>
<td>red1</td>
<td>red2</td>
<td>red3</td>
<td>red4</td>
</tr>
<tr>
<td>rosybrown</td>
<td>rosybrown1</td>
<td>rosybrown2</td>
<td>rosybrown3</td>
</tr>
<tr>
<td>rosybrown4</td>
<td>royalblue</td>
<td>royalblue1</td>
<td>royalblue2</td>
</tr>
<tr>
<td>royalblue3</td>
<td>royalblue4</td>
<td>saddlebrown</td>
<td>salmon</td>
</tr>
<tr>
<td>salmon1</td>
<td>salmon2</td>
<td>salmon3</td>
<td>salmon4</td>
</tr>
<tr>
<td>sandybrown</td>
<td>seagreen</td>
<td>seagreen1</td>
<td>seagreen2</td>
</tr>
<tr>
<td>seagreen3</td>
<td>seagreen4</td>
<td>seashell</td>
<td>seashell1</td>
</tr>
<tr>
<td>seashell2</td>
<td>seashell3</td>
<td>seashell4</td>
<td>sienna</td>
</tr>
<tr>
<td>sienna1</td>
<td>sienna2</td>
<td>sienna3</td>
<td>sienna4</td>
</tr>
<tr>
<td>skyblue</td>
<td>skyblue1</td>
<td>skyblue2</td>
<td>skyblue3</td>
</tr>
<tr>
<td>skyblue4</td>
<td>slateblue</td>
<td>slateblue1</td>
<td>slateblue2</td>
</tr>
<tr>
<td>slateblue3</td>
<td>slateblue4</td>
<td>slategray</td>
<td>slategray1</td>
</tr>
<tr>
<td>slategray2</td>
<td>slategray3</td>
<td>slategray4</td>
<td>slategrey</td>
</tr>
<tr>
<td>snow</td>
<td>snow1</td>
<td>snow2</td>
<td>snow3</td>
</tr>
<tr>
<td>snow4</td>
<td>springgreen</td>
<td>springgreen1</td>
<td>springgreen2</td>
</tr>
<tr>
<td>springgreen3</td>
<td>springgreen4</td>
<td>steelblue</td>
<td>steelblue1</td>
</tr>
</tbody>
</table>
As the international software market grows, the need to remove the bias toward English in software also grows. International users must be able to interact with an application in their native language. Hard-coded English messages and cultural biases can severely restrict an application’s usefulness.

To address the problem of internationalization, TME 10 AEF supports a message catalog system. All program text (messages, errors, and dialog labels) is stored in separate modules, one for each supported language. When a message is displayed, the application shows the message appropriate to the user’s locale.

A locale specifies the language, character set, collation order, translated messages, monetary formats, and date formats that define a language environment. For example, your installation might require an English locale, a French locale, and a German locale.

The only locale that is guaranteed to be available is the special C locale. The C locale, defined as part of the ANSI C standard, is comprised of English messages in 7-bit ASCII.

---

**Internationalization Support in DSL**

<table>
<thead>
<tr>
<th>steelblue2</th>
<th>steelblue3</th>
<th>steelblue4</th>
<th>tan</th>
</tr>
</thead>
<tbody>
<tr>
<td>tan1</td>
<td>tan2</td>
<td>tan3</td>
<td>tan4</td>
</tr>
<tr>
<td>thistle</td>
<td>thistle1</td>
<td>thistle2</td>
<td>thistle3</td>
</tr>
<tr>
<td>thistle4</td>
<td>tomato</td>
<td>tomato1</td>
<td>tomato2</td>
</tr>
<tr>
<td>tomato3</td>
<td>tomato4</td>
<td>turquoise</td>
<td>turquoise1</td>
</tr>
<tr>
<td>turquoise2</td>
<td>turquoise3</td>
<td>turquoise4</td>
<td>violet</td>
</tr>
<tr>
<td>violetred</td>
<td>violetred1</td>
<td>violetred2</td>
<td>violetred3</td>
</tr>
<tr>
<td>violetred4</td>
<td>wheat</td>
<td>wheat1</td>
<td>wheat2</td>
</tr>
<tr>
<td>wheat3</td>
<td>wheat4</td>
<td>white</td>
<td>whitesmoke</td>
</tr>
<tr>
<td>yellow</td>
<td>yellow1</td>
<td>yellow2</td>
<td>yellow3</td>
</tr>
</tbody>
</table>
Messages and Message Catalogs

Message catalogs are binary files generated by the `gencat` utility from a text source that you create. The message source file generally has the extension `.msg`.

A message source file contains a group of numbered messages. Each catalog is divided into `sets`, which are groups of logically related messages. Although there can be many sets in a catalog, TME 10 applications generally only use two sets. Set number one contains default messages. Set number two contains verbose versions of the messages in set number one. Sets are defined using the `set` directive.

```
$ set 1
message1 ...
messageN
$ set 2
message1 ...
messageN
```

Each message entry in a catalog has the following format:

```
$ key=keyname
n message_body
```

The `keyname` field is a unique key name that can be used to access the message. The `n` field is a message number. The message number must be unique within a set.

A body of a message is comprised of text and `message variables`. Message variables act as placeholders for data that is supplied at run-time. For example, you might have a message entry that looks like:

```
$ key=UserExists
1 User %1$s has not been installed on node %2$s.
```

In the example above, the format directives `%1$s` and `%2$s` are placeholders for a user name and a managed node name.

To create a binary catalog from your message source file, use the `gencat` utility. The syntax of `gencat` is shown below.

```
# gencat catfile source
```
The `catfile` parameter is the name of the output file and `source` is the name of the message source file. By convention, all catalog binaries should have a `.cat` extension.

If `catfile` does not exist, `gencat` creates it. If `catfile` exists, `gencat` merges the source file into the existing binary, replacing duplicate message numbers with the new version.

The final step in creating a message catalog is to install the binary catalogs in the TME 10 message catalog directory. All TME 10 message catalogs are located in the directory `install_dir/interp_type/msg_cat` where `install_dir` is the TME 10 installation directory, and `interp_type` is one of the following:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Machine Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 3.2.5</td>
<td>aix3-r2</td>
</tr>
<tr>
<td>AIX 4.1</td>
<td>aix4-r1</td>
</tr>
<tr>
<td>Data General</td>
<td>dgux5</td>
</tr>
<tr>
<td>HP-UX 9.x</td>
<td>hpux9</td>
</tr>
<tr>
<td>HP-UX 10</td>
<td>hpux10</td>
</tr>
<tr>
<td>Motorola</td>
<td>sysv4-m88k</td>
</tr>
<tr>
<td>NCR (AT&amp;T)</td>
<td>sysv4-att</td>
</tr>
<tr>
<td>Solaris</td>
<td>solaris2</td>
</tr>
<tr>
<td>SunOS</td>
<td>sunos4</td>
</tr>
<tr>
<td>Unixware</td>
<td>uw2-ix86</td>
</tr>
</tbody>
</table>

There is a `msg_cat` directory on the TME 10 server of each TMR. Under the `msg_cat` directory there is a sub-directory for each locale supported by the TMR. For example, the directory `msg_cat/C` contains all the C-locale message catalogs for the TMR.

Copy the binary versions of your catalog into the appropriate locale sub-directories. Although it can be useful to give each version of the catalog a unique name during translation, you must rename them when
installing the binaries; the binary catalogs must have the same name in all locales.

Using Message Catalogs with DSL

DSL provides access to message catalogs with the \texttt{Msg} directive, which has the following syntax:

\texttt{Attribute=Msg("catalog", "default_text", "key", [parameters])}

The following list describes each variable:

- \texttt{Attribute}—one of the following gadget attributes:
  - \texttt{AppendText}
  - \texttt{Choices}
  - \texttt{CopyIn}
  - \texttt{HelpMessage}
  - \texttt{Mnemonic}
  - \texttt{StatusMessage}
  - \texttt{Title}
  - \texttt{Value}

  The specified message will be provided as a gadget attribute.

- \texttt{catalog}—the name of the message catalog that contains the message to be displayed

- \texttt{default_text}—a string that is displayed if the message catalog cannot be found

- \texttt{key} —the key word that identifies the message to be displayed

- \texttt{parameters}—additional arguments that specify additional information at run-time

Consider the following DSL code fragment:

```bash
1   Command Dialog{
2     Variables{
3       String user_name;
4       String host_name;
```
Creating Dialog Descriptors

5   }
6   Text{
7     Name=Warning;
8     Value=Msg("host.cat", "User %1$s has not been
9       installed on node %2$s",
10     "user_not_installed"
11     $user_name, $host_name);

Lines 2 through 5 declare two variables used by the Msg directive: user_name, and host_name. Lines 8 through 11 contain a Value attribute that specifies the Msg directive. The first parameter specifies the message catalog as host.cat.

The second parameter contains a quoted string that is displayed if the specified message catalog cannot be found.

The third parameter is the key of a particular message in the host.cat catalog, in this case user_not_found.

The last two parameters ($user_name and $host_name) specify the value of the replaceable parameters %1$s and %2$s. The actual values of $user_name and $host_name are determined at run-time by a callback method you provide.

Creating Dialog Descriptors

After you create a DSL source file, you must create a binary descriptor for the dialog. To do this, you must run the source file through the DSL compiler. The syntax for the DSL compiler is shown below.


The command line switches enable the DSL compiler to execute instructions prior to or during compilation. This provides a flexible compiling scheme for your DSL files.

The –D option defines a symbol for use by a preprocessor. The option defines the symbol name to have the value def. If def is omitted, name is defined to have a null value. The –D option is valid only if –P or –E is specified. It may be given multiple times.
The \texttt{-E} option instructs the DSL compiler to run its input through the preprocessor specified by \texttt{path} but not to compile the result. The preprocessor output is written to the output file specified by the \texttt{-o} option, or to standard output if the \texttt{-o} option is omitted. The DSL compiler automatically defines the symbol \texttt{dsl} for the preprocessor when the \texttt{-E} option is included. If \texttt{path} is omitted, \texttt{path} defaults to \texttt{/lib/cpp}. The \texttt{-E} cannot be used with \texttt{-P}.

The \texttt{-P} option causes the DSL compiler to run the input file through the preprocessor specified by \texttt{path}, prior to compilation. If omitted, \texttt{path} defaults to \texttt{/lib/cpp}. The DSL compiler automatically defines the symbol \texttt{dsl} for the preprocessor when \texttt{-P} is specified.

The \texttt{-u} option prevents the DSL compiler from pre-defining the \texttt{dsl} symbol when the \texttt{-P} or \texttt{-E} switches are specified.

The \texttt{-e} option suppresses informative messages, warning messages, and error messages.

The \texttt{-I} option specifies the path of a directory to be searched for files specified by the \texttt{#include} directive in the DSL file. This parameter is passed to the specified preprocessor and may be given multiple times. It is valid only if the \texttt{-E} or \texttt{-P} option is specified.

The \texttt{-help} message causes the DSL compiler to print a short usage message and exit without compiling or preprocessing.

The \texttt{-o} option specifies the name of an output file. If omitted, the DSL compiler writes to standard output.

The \texttt{-V} option causes the DSL compiler to print its version number and exit without compiling or preprocessing. If this option is present, all other options are ignored.

The \texttt{sourcefile} parameter is the name of the file to be compiled. If \texttt{sourcefile} does not exist, the DSL compiler will attempt to compile \texttt{sourcefile.dsl} instead (the file named by \texttt{sourcefile} with a \texttt{.dsl} extension). If the \texttt{sourcefile} parameter is omitted, the DSL compiler reads from standard input.
Creating Dialog Descriptors

## DSL Preprocessor Directives

The DSL compiler uses the `cpp` preprocessor and therefore supports preprocessing directives if you invoke the compiler with the `-P` or `-E` flags. The following directives are supported:

- **#include** — Permits the inclusion of other files in your main DSL source files.
- **#define** — Permits the use of symbols instead of hard-coded names and constants.
- **#if** — Permits the use of conditional compilation.

See the `dsl` manual page for more information on using the `cpp` preprocessor with the DSL compiler.

## Previewing Dialogs

A DSL dialog previewer is available for previewing dialogs before installation. To use the previewer you must first compile the dialog using the DSL compiler.

After compiling the dialog, you can preview it by invoking the `tivoli` command using the `-preview` switch as shown below.

```
tivoli -preview filename.d
```

In this example, `filename` is the name of a compiled dialog descriptor. If the DSL file contains any parameter-valued dialog variables, you must specify them when you run the `tivoli` command.

```
tivoli -preview filename.d var-1 value-1 var2 value-2 ...
```

Consider the following example dialog, which is contained in a file named `main.dsl`:

```plaintext
1 Command Dialog {
2 Variables {
3  String t;
4  String mess;
5 }
6 Attributes {
7  Name = main;
8  Title = $t;
```
Creating Dialog Descriptors

```plaintext
9 }
10 Gadgets {
11  Message {
12     Name = m;
13     Value = $mess;
14 }
15 }
16 }
```

The command shown below previews the dialog. The values of the `t` and `mess` variables are passed on the command line.

```plaintext
tivoli -preview main.d t "Here is a title" mess "This is a message"
```

If the dialog contains a variable of type `CString` and contains blanks or other special characters, delimit the string with apostrophes. For example, if the variables in the example dialog were of type `CString`, you should invoke `tivoli` as shown:

```plaintext
tivoli -preview main.d t "'Here is a title'" mess "'This is a message'"
```

See the `tivoli` manual page for more information about this feature.

Reverse Compiling Dialogs

A DSL reverse compiler is available that enables you to convert a binary dialog descriptor to an ASCII DSL file. The command for reverse compiling dialog descriptors is `rdsl`. The syntax of the `rdsl` command is shown below.

```plaintext
rdsl filename.d > filename.dsl
```

Although the `rdsl` program converts a compiled dialog to DSL source code, it is not identical to the original. The following differences exist from the reverse compiled version to the original:

- The generated version does not retain the comments included in the original.
- The spacing can be different.
- The ordering of the attributes can be different.
An Example: Designing a Dialog

This section presents an example dialog for use in a fictitious food order-out program. It presents the dialog, the DSL source code fragments that produce the dialog, and an explanation of the dialog component choices made by the designer.

The program allows a user to order food directly from a workstation. The dialog collects the following information about the food and the user:

- User name.
- The desired meal — breakfast, lunch, and so on.
- The intensity of the user’s hunger.
- Food type — Thai, American, and so on.
- Method of payment.

This example only describes a dialog’s design and implementation in DSL; it does not include any discussion of the callback methods required to make a functioning application.

Our example dialog looks like this when it is complete:
Determining Gadgets Types

Sometimes it is a good idea to sketch the dialog before writing the DSL to define it. This step helps identify the required gadgets and their location in the dialog. After you sketch the dialog, you can identify the types of gadgets the dialog will need. The table below shows the gadget required for each component of the example dialog.

<table>
<thead>
<tr>
<th>Component</th>
<th>Gadget</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name</td>
<td>Text</td>
</tr>
<tr>
<td>Hunger level</td>
<td>Choice</td>
</tr>
<tr>
<td>Meal</td>
<td>Choice</td>
</tr>
<tr>
<td>Food type</td>
<td>Choice</td>
</tr>
<tr>
<td>Payment</td>
<td>Switch</td>
</tr>
<tr>
<td>Apply</td>
<td>CommandButton</td>
</tr>
<tr>
<td>Dismiss</td>
<td>CommandButton</td>
</tr>
</tbody>
</table>

Beginning the Dialog Specification

The first line of a DSL file defines the dialog type, in this case command dialog. Since every dialog must have a Name attribute, the dialog must also contain an Attributes block. The DSL fragment below begins the dialog and defines its attributes.

```plaintext
1 Command Dialog {
2   Attributes {
3     Name = sample_dialog;
4     Title = "Determine Meal Parameters";
5   }
```

The Attributes block also defines a Title attribute for the dialog. The value of the Title attribute is displayed at the top of the dialog.

Specifying the Dialog Gadgets

Each command dialog has a Gadgets specification that describes the gadgets of the dialog. By default, the gadgets are laid out vertically, top...
An Example: Designing a Dialog

to bottom. The first gadget in this section is the **Text** gadget, which collects the user’s name. The DSL fragment is shown below.

```plaintext
Gadgets
{
  Text
  {
    Name = consumer;
    Title = "Name:";
  }
}
```

By default, DSL places the **Text** gadget title to the left of the gadget, which is an acceptable place for it in the example application.

A **Choice** gadget follows the **Text** gadget. The values assigned to the **Choices** attribute of this gadget compose a special value known as a **choice list**. The attribute is comprised of a comma-separated item list. The Desktop displays the quoted string in the first part of each item to the user. When a user selects an item, the Desktop returns the value of the quoted string inside the curly braces to any callbacks that reference the **Choice** gadget’s value. The gadget’s DSL fragment is shown below.

```plaintext
Choice
{
  Name = meal;
  Title = "Meal";
  Show = ONE;
  Choices =
    "Breakfast" {"Breakfast"},
    "Brunch"    {"Brunch"},
    "Lunch"     {"Lunch"},
    "Snack"     {"Snack"},
    "Dinner"    {"Dinner"};
}
```

The **Show** attribute dictates how many of the choices are shown to the user at once. For this gadget, **Show** is set to **ONE**. This produces the choice list below.

![Choice list with options: Breakfast, Brunch, Lunch, Snack, Dinner]
An Example: Designing a Dialog

Next, we will add the two choice gadgets side by side on the dialog. These are the hunger level and meal type components. Together, these two gadgets form a group that is aligned horizontally on the dialog. The DSL code below defines the group that contains the two gadgets.

```plaintext
Group {
  Attributes {
    Name = g0;
    Layout = HORIZONTAL;
    TitlePos = TOP;
    Border = YES;
  }
  Gadgets {
    Choice {
      Name = hunger_level;
      Title = "Level of Hunger";
      TitlePos = TOP;
      Show = ALL;
      Layout = VERTICAL;
      Choices =
        "NOT AT ALL" {"NOT AT ALL"},
        "SOME" {"SOME"},
        "VERY" {"VERY"};
    }
    Choice {
      Name = food;
      Title = "Type of Meal";
      TitlePos = TOP;
      Show = SOME;
      Sort = YES;
      Choices =
        "Italian" {"Italian"},
        "Thai" {"Thai"},
        "Chinese" {"Chinese"},
        "French" {"French"},
        "African" {"African"},
        "American" {"American"},
        "German" {"German"},
        "Mongolian" {"Mongolian"},
        "Japanese" {"Japanese"},
        "Russian" {"Russian"},
        "Mexican" {"Mexican"},
    }
  }
}
```
An Example: Designing a Dialog

```
38      "Basque" {"Basque"},
39      "Spanish" {"Spanish"};
40    }
41  }
42 )

By default the title of a gadget is shown to the left of the gadget. The TitlePos attributes on lines 12 and 23 change the title position so that it is displayed above the gadget.

The first gadget, `hunger_level`, has its Show attribute set to all. This causes the list to be displayed as a set of radio buttons. Line 14 sets the Layout attribute to vertical, which arranges the buttons into a column. The result is shown below.

Level of Hunger

- NOT AT ALL
- SOME
- VERY

The Show attribute of the second gadget is set to SOME. This causes the gadget to appear as a scrolling choice list, like the one shown below.

<table>
<thead>
<tr>
<th>❯ African</th>
<th>❯ American</th>
<th>❯ Basque</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>❯ Chinese</td>
</tr>
</tbody>
</table>
At this point, we have added four gadgets to our dialog. The figure below shows what the dialog looks like so far.

The **Group** gadget `g0`, which contains the **hunger_level** and **food** gadgets, is followed by a single **Message** gadget. This gadget provides a title for the gadgets defined by subsequent DSL code. The following code fragment shows the **Message** gadget:

```plaintext
Message {
    Name = payment;
    Title = "Method of Payment";
}
```

For this example, we will make the **Method of Payment** gadget a set of switches. A switch is a two-position indicator. Unlike radio buttons, switches are not mutually exclusive. So, for this example, the user can select more than one method of payment.

The **Method of Payment** gadget is defined in DSL as a group, called `g1`. The `g1` group, which is laid out horizontally, is comprised of two
other groups (g2 and g3), which are laid out vertically. This forms two columns of payment switches.

The DSL for the Method of Payment component is shown below.

```
1 Group {
2  Attributes {
3    Name = g1;
4    Layout = HORIZONTAL;
5  }
6  Gadgets {
7    Group {
8      Attributes {
9        Name = g2;
10       Layout = VERTICAL;
11       Alignment = LEFT;
12    }
13    Gadgets {
14      Switch {
15        Name = cash;
16        Title = "Cash/Check";
17        On = "ENABLED";
18        Off = "DISABLED";
19        Value = "ENABLED";
20      }
21    }
22    }
23    Switch {
24      Name = visa;
25      Title = "VISA/Mastercard";
26      On = "ENABLED";
27      Off = "DISABLED";
28    }
29  }
30}
```
The first switch gadget, cash is defined on lines 14–20. The **On** and **Off** attributes define the gadget’s value when it is selected and not selected, respectively. Because cash is the preferred payment method, it is the default choice. Its **Value** attribute is set to **ENABLED** (line 19) so that it appears in the selected state when the dialog displays. The remaining switch gadgets are defined in a similar manner.

Finally, the program specifies two **CommandButton** gadgets: **Apply** and **Dismiss**. If the user selects **Apply**, the program invokes a callback to place the order. If the user selects **Dismiss**, the program dismisses the dialog. The code fragment below defines these two buttons.
An Example: Designing a Dialog

The final product is shown below, followed by a complete listing of the DSL source code.

![Diagram of the dialog interface]

Command Button {
  Name = dismiss;
  Title = "Dismiss";
  Commands = dtt_dismiss($owner, sample_dialog, $self);
}

Command Dialog {
  Attributes {
    Name = sample_dialog;
    Title = "Determine Meal Parameters";
  }
  Gadgets {
    Text {
      Name = consumer;
      Title = "Name:";
    }
  }
}
An Example: Designing a Dialog

Dialog Specification

Language

Choice {
  Name = meal;
  Title = "Meal";
  Show = ONE;
  Choices =
    "Breakfast" {"Breakfast"},
    "Brunch" {"Brunch"},
    "Lunch" {"Lunch"},
    "Snack" {"Snack"},
    "Dinner" {"Dinner"};
}

Group {
  Attributes {
    Name = g0;
    Layout = HORIZONTAL;
    TitlePos = TOP;
    Border = YES;
  }

  Gadgets {
    Choice {
      Name = hunger_level;
      Title = "Level of Hunger";
      TitlePos = TOP;
      Show = ALL;
      Layout = VERTICAL;
      Choices =
        "NOT AT ALL" {"NOT AT ALL"},
        "SOME" {"SOME"},
        "VERY" {"VERY"};
    }

    Choice {
      Name = food;
      Title = "Type of Meal";
      TitlePos = TOP;
      Show = SOME;
      Sort = YES;
      Choices =
        "Italian" {"Italian"},
        "Thai" {"Thai"},
    }
  }
}
An Example: Designing a Dialog

```
56   "Chinese"{"Chinese"},
57   "French"{"French"},
58   "African"{"African"},
59   "American"{"American"},
60   "German"{"German"},
61   "Mongolian"{"Mongolian"},
62   "Japanese"{"Japanese"},
63   "Russian"{"Russian"},
64   "Mexican"{"Mexican"},
65   "Basque"{"Basque"},
66   "Spanish"{"Spanish"};
67   }
68   }
69   }
70   }
71   Message {
72       Name = payment;
73       Title = "Method of Payment";
74   }
75   }
76   Group {
77       Attributes {
78           Name = g1;
79           Layout = HORIZONTAL;
80       }
81   }
82   Gadgets {
83       Group {
84           Attributes {
85           Name = g2;
86           Layout = VERTICAL;
87           Alignment = LEFT;
88           }
89       }
90       Gadgets {
91       Switch {
92           Name = cash;
93           Title = "Cash/Check";
94           On = "ENABLED";
95           Off = "DISABLED";
96           Value = "ENABLED";
97           }
98       Switch {
99       }
```
An Example: Designing a Dialog

```
98           Name = visa;
99           Title = "VISA/Mastercard";
100          On = "ENABLED";
101          Off = "DISABLED";
102      }
103      }
104     }
105
106     Group {  
107         Attributes {  
108             Name = g3;  
109             Layout = VERTICAL;  
110             Alignment = LEFT;  
111         }  
112         Gadgets {  
113             Switch {  
114                 Name = amex;  
115                 Title = "American Express";  
116                 On = "ENABLED";  
117             }  
118             Switch {  
119                 Name = discover;  
120                 Title = "Discover";  
121                 On = "ENABLED";  
122                 Off = "DISABLED";  
123             }  
124         }  
125     }  
126 }
127 }
128 }
129
130     CommandButton {  
131         Name = apply;  
132         Title = "Apply";  
133         Commands = place_order($g0.hunger_level,  
134             $g0.food),  
135             dtc_dismiss($owner, sample_dialog,  
136             $self);  
137 }
138 }
139 CommandButton {
```
An Example: Designing a Dialog

```
140    Name = dismiss;
141    Title = "Dismiss";
142    Commands = dtc_dismiss($owner, sample_dialog,
143                                    $self);
144   }
145 }
146 }
```

Adjusting the Dialog

Looking at the new dialog, you probably noticed several aesthetic problems with it. The two choice gadgets in the center are too close together, and the top margin of the Method of Payment line is too small. So, the dialog layout needs some adjustment.

You can adjust the dialog layout by inserting empty Message gadgets. In the example dialog, we will add three message gadgets between the hunger_level and food choice gadgets.

```
1 Message
2 {
3    Name = for_layout1;
4 }
5 Message
6 {
7    Name = for_layout2;
8 }
9 Message
10 {
11    Name = for_layout3;
12 }
```

The Message gadgets provide blank space between the two gadgets. Adding a final Message gadget after the g0 group increases the top
margin of the **Method of Payment** component. The revised dialog is shown below

![Diagram of the revised dialog](image)

**Grid Alignment**

In our example dialog, the gadgets were arranged on the dialog by placing them in groups and using blank **Message** gadgets as spacers. You could also use the grid alignment attributes to position the gadgets within the dialog. Using grid alignments simplifies the dialog’s DSL code and eliminates the need for adjusting the dialog after you create it.

DSL provides the following grid alignment attributes:

- **GridWidth**
- **GridHeight**
- **GridHorizontal**
- **GridVertical**
- **ChildColumnAlignment**
- **ChildRowAlignment**
To use the grid alignment attributes, you must set the gadget’s Layout attribute to GRID. The following code fragment specifies some of the gadgets from the previous example using grid alignment attributes rather than groups of gadgets.

```plaintext
Group
{
  Attributes
  {
    Name = g0;
    Layout = GRID;
    TitlePos = TOP;
    Border = YES;
    GridWidth = 10;
    GridHeight = 8;
  }
  Gadgets
  {
    Choice
    {
      Name = hunger_level;
      Title = "Level of Hunger";
      TitlePos = TOP;
      Show = ALL;
      GridHorizontal = 1;
      GridVertical = 1;
      ChildColumnAlignment = CENTER;
      ChildRowAlignment = CENTER;
      Choices =
        "NOT AT ALL" {"NOT AT ALL"},
        "SOME" {"SOME"},
        "VERY" {"VERY"};
    }
    Choice
    {
      Name = food;
      Title = "Type of Meal";
      TitlePos = TOP;
      Show = SOME;
      Sort = YES;
      GridHorizontal = 7;
      GridVertical = 1;
    }
  }
}
```
An Example: Designing a Dialog

---

```plaintext
ChildColumnAlignment = CENTER;
Choices =
  "Italian" {"Italian"},
  "Thai" {"Thai"},
  "Chinese" {"Chinese"},
  "French" {"French"},
  "African" {"African"},
  "American" {"American"},
  "German" {"German"},
  "Mongolian" {"Mongolian"},
  "Japanese" {"Japanese"},
  "Russian" {"Russian"},
  "Mexican" {"Mexican"},
  "Basque" {"Basque"},
  "Spanish" {"Spanish"};
Message {
  Name = payment;
  Title = "Method of Payment";
  TitlePos = CENTER;
}
Group {
  Attributes {
    Name = g1;
    Layout = GRID;
    TitlePos = CENTER;
    Border = NO;
    GridWidth = 10;
    GridHeight = 8;
  }
  Gadgets {
    Switch {
      Name = cash;
      Title = "Cash/Check";
      GridHorizontal = 1;
      GridVertical = 1;
      ChildColumnAlignment = RIGHT;
      ChildRowAlignment = CENTER;
      On = "ENABLED";
      Off = "DISABLED";
      Value = "ENABLED";
    }
  }
}
```
An Example: Designing a Dialog

```java
80    Switch {
81       Name = visa;
82       Title = "VISA/Mastercard";
83          GridHorizontal = 1;
84          GridVertical = 2;
85          ChildColumnAlignment = RIGHT;
86          ChildRowAlignment = CENTER;
87       On = "ENABLED";
88       Off = "DISABLED";
89    }
90    Switch {
91       Name = amex;
92       Title = "American Express";
93          GridHorizontal = 6;
94          GridVertical = 1;
95          ChildColumnAlignment = RIGHT;
96          ChildRowAlignment = CENTER;
97       On = "ENABLED";
98       Off = "DISABLED";
99    }
100   Switch {
101       Name = discover;
102       Title = "Discover";
103          GridHorizontal = 6;
104          GridVertical = 2;
105          ChildColumnAlignment = RIGHT;
106          ChildRowAlignment = CENTER;
107       On = "ENABLED";
108       Off = "DISABLED";
109    }
110 } 111 }
112 }

The `GridHeight` and `GridWidth` attributes specify a grid on the dialog. Subsequent grid attributes are specified relative to the upper left-hand corner of the grid. The `GridHorizontal` and `GridVertical` attributes place a gadget within the grid.

After placing the gadget, you can use the `ChildColumnAlignment` and `ChildRowAlignment` to position the gadget within its grid section. The `ChildColumnAlignment` attribute aligns gadgets `LEFT`,
**An Example: Designing a Dialog**

**Language**

**Dialog Specification**

Rights, or CENTER within its grid section. Similarly, ChildRowAlignment aligns gadgets TOP, BOTTOM, or CENTER. In addition to these values ChildColumnAlignment and ChildRowAlignment support one other value: STRETCH. The STRETCH values causes the gadget to be expanded to cover the entire grid section vertically or horizontally. Thus, the dialog specification for the hunger_level choice gadget can be modified as shown below.

```cpp
1 Choice
2 {
3     Name = hunger_level;
4     Title = "Level of Hunger";
5     TitlePos = TOP;
6     Show = ALL;
7         GridHorizontal = 1;
8         GridVertical = 1;
9         ChildColumnAlignment = STRETCH;
10        ChildRowAlignment = CENTER;
11       Choices =
12             "NOT AT ALL" {"NOT AT ALL"},
13             "SOME" {"SOME"},
14             "VERY" {"VERY"};
15 }
```

This modification expands the gadget horizontally to the full width of the grid section. Similarly, the specification could be modified as shown:

```cpp
1 Choice
2 {
3     Name = hunger_level;
4     Title = "Level of Hunger";
5     TitlePos = TOP;
6     Show = ALL;
7         GridHorizontal = 1;
8         GridVertical = 1;
9         ChildColumnAlignment = CENTER;
10        ChildRowAlignment = STRETCH;
11       Choices =
12             "NOT AT ALL" {"NOT AT ALL"},
13             "SOME" {"SOME"},
14             "VERY" {"VERY"};
```
This modification expands the gadget vertically to the full height of the grid section.
Custom Dialogs

Introduction

Nearly all customizations with TME 10 AEF involve adding a gadget to a dialog or a command to a menu. Dialogs and menus are stored as binary *dialog descriptors*, which are created with the DSL compiler. When you customize a dialog, you create a new version of the dialog descriptor and install it on a resource or instance.
To understand how dialogs and customizations are organized in TME10, look at the figure below.

As you can see, a resource’s Tivoli-supplied dialogs—except those associated with creation of instances—are stored in the resource’s user interface component. All instances of the resource use these dialogs. The resource also contains an extension component, which is where custom dialogs are stored. The custom dialogs override the corresponding original dialog. Custom dialogs stored in the resource’s extension area affect all instances of the resource.

In addition to the resource’s extension component, each instance of a resource contains an extension component. Custom dialogs on an instance affect only that instance.

Dialogs associated with the creation of instances are not stored with the resource. Since create dialogs must exist independently of the instances they create, they are stored with the resource’s default
policies. Each default policy has an extension component that allows you to customize the create dialog. You can create custom dialogs that affect all default policies or individual policies.

To customize a dialog, you must

- Retrieve the dialog descriptor.
- Reverse compile the descriptor.
- Edit the DSL source code to add your custom gadget.
- Compile a new dialog descriptor.
- Install the custom descriptor.

This chapter discusses all the above steps, except editing the DSL source code. If you are unfamiliar with DSL, you should read Chapter 4, “Dialog Specification Language” before proceeding with this chapter.

If you are customizing a profile-based application, there are a few considerations. Chapter 10, “TME 10 AEF and Profiles” covers how to customize a profile-based resource’s dialogs.

**Retrieving a Dialog Descriptor**

The `wgetdialog` command retrieves the binary dialog descriptor of a dialog. You can then use the `rdsl` reverse compiler to get the DSL source code for the dialog. See “Reverse Compiling a Descriptor” on page 5-6 for more information about converting a dialog descriptor into DSL source code.

You can use the `wgetdialog` command to retrieve descriptors for

- Custom dialogs installed for an entire resource.
- Original resource-wide dialogs.
- Custom dialogs installed on an instance of a resource.
- Custom create dialogs installed across all default policies.
- Original create dialogs.
- Custom create dialogs installed on a single default policy.
Retrieving a Dialog Descriptor

**Getting a Resource-Wide Dialog**

The `wgetdialog` syntax for retrieving a dialog used by all instances of a resource is shown below.

```
wgetdialog -r resource [-s] [-t tmr] dialog-name > dialog
```

The `-r` option specifies the resource type that uses the dialog. The `dialog-name` parameter is the name of the dialog to retrieve. Refer to Appendix A, “Dialog Descriptors” for a complete list of TMF dialog names.

If there is a custom version of the dialog for the resource, `wgetdialog` returns it. Otherwise, `wgetdialog` returns the original version of the dialog. The command writes the descriptor to standard output. To save the descriptor to a file, you must redirect the command’s standard output to a file.

```
% wgetdialog -r ManagedNode add_dialog > add.d
```

The example above retrieves the `add_dialog` descriptor for the `ManagedNode` resource and writes the DSL source code to the file `add.d`.

By default, `wgetdialog` works in the TMR where the command is invoked. However, you can use the `-t` option to get a dialog from a remote TMRs as well. The `-t` option takes a parameter, which is the name of the remote TMR.

```
% wgetdialog -r ManagedNode -t Europe add_dialog > add.d
```

The example above is similar to the previous example, except it retrieves the dialog from the `ManagedNode` resource type in the `Europe` TMR.

The `-s` option allows you to retrieve the original version of a dialog even if there is a custom version installed. For example, if there is a customer version of the `add_dialog` dialog installed, you could use the following command to retrieve the original version:

```
% wgetdialog -s -r ManagedNode add_dialog > add.d
```
Getting a Per-Instance Custom Dialog

You can also retrieve a custom dialog from an instance of a resource type. The `wgetdialog` syntax for retrieving a per-instance customization is shown below.

```
wgetdialog -l label [-t tmr] dialog-name > dialog
```

The syntax is similar to the syntax for getting a resource dialog, except that you must use the `-l` option to specify the object path of the instance. You also do not need to specify the resource type with the `-r` option, since the type must be embedded in the object path.

If there is no custom version of the dialog on the instance, `wgetdialog` exits with an error. The command writes the descriptor to standard output. To save the descriptor to a file, you must redirect the command’s standard output to a file.

```
% wgetdialog -l @ManagedNode:brice add_dialog > add.d
```

The example above retrieves the custom version of the `add_dialog` dialog for the `brice` managed node.

Getting a Create Dialog

Dialogs used to create instances of a resource are stored differently from other dialogs. This is because create dialogs must exist independently of the instances they create. So, create dialogs are stored with the resource’s default policies. TME 10 AEF allows you to customize create dialogs on a resource-wide and policy-specific basis.

To get the descriptor of a create dialog, you must use the `-c` or `-C` options of the `wgetdialog` command. The `wgetdialog` syntax for retrieving a create dialog is shown below.

```
wgetdialog { -c policy | -C } -r resource [-t tmr] dialog-name > dialog
```

The `-C` option retrieves a create dialog for a specific resource type. If the create dialog has been customized for all default policies of the resource, `wgetdialog` returns the custom version. Otherwise, `wgetdialog` returns the original version.

```
% wgetdialog -C -r Administrator create_admin > create.d
```
Reverse Compiling a Descriptor

In the example above, `wgetdialog` first looks for a custom version of the `create_admin` dialog. If a custom version does not exist, `wgetdialog` returns the original version of `create_admin`.

The `-c` option retrieves a create dialog from a specific default policy. If there is an individual customization for the dialog, `wgetdialog` returns it. If there is not an individual customization, `wgetdialog` looks for a version customized for all default policies. If there is no resource-wide customization, `wgetdialog` returns the original version.

```bash
% wgetdialog -c BasicTaskLibrary -r TaskLibrary \    
    make_task_library > make_tl.d
```

In the example above, `wgetdialog` searches the `BasicTaskLibrary` default policy for a custom version of the `make_task_library` dialog. If the policy does not have an individual version of the dialog, `wgetdialog` checks for a version customized for all default policies. If there is no resource-wide customization, `wgetdialog` returns the original version of the dialog.

Reverse Compiling a Descriptor

After you have retrieved a dialog descriptor with `wgetdialog`, you must reverse compile it to get the dialog’s DSL source code. The command for reverse compiling dialog descriptors is `rdsl`. The syntax of the `rdsl` command is shown below.

```bash
rdsl filename.d > filename.dsl
```

Although the `rdsl` program converts a compiled dialog to DSL source code, it is not identical to the original. The following differences exist from the reverse compiled version to the original:

- The generated version does not retain the comments included in the original.
- The spacing can be different.
- The ordering of the attributes can be different.

An example of the `rdsl` command is shown below.

```bash
% rdsl add.d > add_dialog.dsl
```
Installing a Custom Dialog

The example reverse compiles the `add.d` descriptor and redirects the result to `add_dialog.dsl`.

You can pipe the output of the wgetdialog directly to rdsl to combine the retrieve/reverse compile steps. An example is shown below.

```
% wgetdialog –r ManagedNode add_dialog | rdsl > add_dialog.dsl
```

The example retrieves the `add_dialog` descriptor and pipes it through the `rdsl` compiler. The result is written to the file `add_dialog.dsl`.

Installing a Custom Dialog

After you have modified the DSL source code and used the DSL compiler to create a new dialog descriptor, you must install your custom dialog. The `wputdialog` command installs a custom dialog.

The command allows you to install a custom dialog for

- An entire resource type.
- Specific instances of a resource type.

If your custom dialog is a create dialog, you can install it on all default policies or on individual policies.

Customizing a Resource Type

The `wputdialog` syntax for installing a resource-wide customization is shown below.

```
wputdialog –r resource [–t tmr... | –T] dialog-name < dialog
```

The `–r` option specifies the resource type that uses the dialog. The `dialog-name` parameter is the name of the dialog to retrieve. Refer to Appendix A, “Dialog Descriptors” for a complete list of TMF dialog names.

The command reads the compiled dialog descriptor from standard input, so you must redirect the descriptor file on the command line.

```
% wputdialog –r Administrator tmr_roles < tmr_roles.d
```
Installing a Custom Dialog

The example above installs a custom version of the Administrator resource’s tmr_roles dialog. The descriptor file, tmr_roles.d, is redirected to the command’s standard input.

By default, the wputdialog command installs the custom dialog in the TMR where the command is invoked. However, you can use the –t option to install the dialog in remote TMRs. The –t option takes a parameter, which is the name of the remote TMR. You can specify the –t option more than once to install the dialog in multiple TMRs.

% wputdialog –t Wasatch –t LaSalle –r Administrator \       tmr_roles < tmr_roles

The example above installs a custom version of the Administrator resource’s tmr_roles dialog. The command installs the dialog in the Wasatch TMR and the LaSalle TMR.

If you want to install the dialog in all connected TMRs, you can use the –T option instead of using the –t option to specify them all individually. When you use the –T option, wputdialog also installs the dialog in the current TMR.

% wputdialog –T –r Administrator tmr_roles < tmr_roles.d

The example above installs the tmr_roles dialog in the current TMR and all remote TMRs.

Customizing an Instance

To install a custom dialog on an individual instance, use the –l option of the wputdialog option. The wputdialog syntax for installing a dialog on an instance is shown below.

wputdialog –l label dialog-name < dialog

The –l option specifies the object path to an instance. You do not have to use the –r option to specify the resource type, since it must be included in the object path.

The command reads the compiled dialog descriptor from standard input, so you must redirect the descriptor file on the command line. An example of per-instance customization is shown below.

% wputdialog –l @Administrator:juan tmr_roles < tmr_roles.d
Removing a Custom Dialog

The example installs a custom version of the `tmr_roles` dialog on the *Administrator* instance *juan*.

### Customizing a Create Dialog

You can use `wputdialog` to install a custom version of a create dialog. You can install a dialog on all default policies of a resource, or on an individual policy. The `wputdialog` syntax is shown below.

```
wputdialog {–C | –c policy} –r resource [–t tmr... | –T] dialog-name < dialog
```

The `–C` option installs a create dialog on all default policies of a resource type. An example is shown below.

```bash
% wputdialog –C –r Administrator create_admin < create.d
```

In the example, `wputdialog` installs a custom version of the `create_admin` dialog on all default policies of the *Administrator* resource.

The `–c` option installs a custom dialog on a specific default policy. An example is shown below.

```bash
% wputdialog –c BasicTaskLibrary –r TaskLibrary \ 
  BasicTaskLibrary < make_tl.d
```

In the example, `wputdialog` installs a custom version of the `make_task_library` dialog on the *BasicTaskLibrary* default policy.

As with its other forms, the `wputdialog` command normally works with default policies in the TMR where the command is run. You can use the `–t` and `–T` options to install custom create dialogs in remote TMRs.

### Removing a Custom Dialog

The `wrmdialog` command removes a custom dialog from a resource or instance. You can only remove dialogs that were added with the `wputdialog` command. The `wrmdialog` command allows you to remove custom dialogs from:
Removing a Custom Dialog

- An entire resource type.
- Specific instances of a resource type.

If the custom dialog is a create dialog, you can remove it from all default policies or from individual policies.

You can only remove a dialog in the same context as it was installed. For example, if you installed the dialog as a resource-wide customization, you must remove it on that basis. You cannot install a resource-wide customization and then remove it from a few instances.

Removing a Resource-Wide Custom Dialog

The `wrmdialog` syntax for removing a resource-wide customization is shown below.

```
wrmdialog -r resource [-t tmr... | -T] dialog-name
```

The `–r` option specifies the resource type that uses the dialog. The `dialog-name` parameter is the name of the dialog to remove. The command only removes customizations. If there is no custom version of the specified dialog, `wrmdialog` exits with an error.

An example of the `wrmdialog` command is shown below.

```
% wrmdialog -r Administrator tmr_roles
```

The example removes a custom version of the `Administrator` resource’s `tmr_roles` dialog.

By default, the `wrmdialog` command works in the TMR where the command is invoked. However, you can use the `–t` option to remove a dialog in remote TMRs. The `–t` option takes a parameter, which is the name of the remote TMR. You can specify the `–t` option more than once to remove the dialog from multiple TMRs.

```
% wrmdialog -t Wasatch -t LaSalle -r Administrator tmr_roles
```

The example above removes the custom version of the `Administrator` resource’s `tmr_roles` dialog in the `Wasatch` and `LaSalle` TMRs.

If you want to remove the customization in all connected TMRs, you can use the `–T` option instead of using the `–t` option to specify each
Removing a Custom Dialog

TMR individually. When you use the `-T` option, `wrmdialog` also removes the dialog in current TMR.

```bash
% wrmdialog -T -r Administrator tmr_roles
```

The example above removes the custom version of the `tmr_roles` dialog in the current TMR and all connected TMRs.

Removing a Per-Instance Custom Dialog

To remove a custom dialog from an individual instance, use the `-l` option of the `wrmdialog` option. The `wrmdialog` syntax for removing a dialog from an instance is shown below.

```bash
wrmdialog -l label dialog-name
```

The `-l` option specifies the object path to an instance. You do not have to use the `-r` option to specify the resource type, since it must be included in the object path.

```bash
% wrmdialog -l @Administrator:juan tmr_roles
```

The example removes the custom version of the `tmr_roles` dialog on the Administrator instance `juan`. If there is not a custom version of the dialog for the specified instance, `wrmdialog` exits with an error.

Removing a Custom Create Dialog

You can use `wrmdialog` to remove a custom create dialog from all default policies or an individual policy. The `wrmdialog` syntax is shown below.

```bash
wrmdialog { -C | -c policy } -r resource [ -t tmr... | -T ] dialog-name
```

The `-C` option removes a create dialog from all default policies of a resource type. An example is shown below.

```bash
% wrmdialog -C -r Administrator create_admin
```

In the example, `wrmdialog` removes the custom version of the `create_admin` dialog from all default policies of the Administrator resource.
The `–c` option removes a custom dialog on a specific default policy. An example is shown below.

```
% wrmdialog –c BasicTaskLibrary –r TaskLibrary \
    make_task_library
```

In the example, `wrmdialog` removes the custom version of the `make_task_library` dialog from the `BasicTaskLibrary` default policy.

As with resource-wide customizations, the `wrmdialog` command normally works with default policies in the TMR where the command is run. You can use the `–t` and `–T` options to remove custom create dialogs from remote TMRs.

### Directory Services for Dialogs

The `wlsdialog` command provides a list of dialogs supported by a resource type. The command also lists the dialogs for which there are custom versions. The syntax for `wlsdialog` is shown below.

```
wlsdialog [-C] -r resource [-t tmr...]  
```

The `–r` option specifies the type of resource. The `–t` option can be used to get a list of dialog from one or more remote TMRs.

The output of `wlsdialog` depends on which language environment (locale) you are using. The output generated in the C locale has the following format:

```
dialog1 (customization status)  
dialog2 name (customization status)  
...  
```

The command lists each dialog name and its status (whether it is the original version or a customization). An example of `wlsdialog` is shown below.

```
% wlsdialog -r TaskLibrary  
dialog name (customization status)  
  
task_output  
task_list  
job_list  
view_dialog  
execute_task (resource-wide customization)  
```
In the example, the `execute_task` and `save_host_and_path` dialogs are resource-wide customizations.

The `wlsdialog` command’s `-C` option provides information about a resource’s create dialogs. An example is shown below.

```bash
% wlsdialog -C -r TaskLibrary
dialog name (customization status)
make_task_library
```

As you can see in the example, the `TaskLibrary` resource has a single create dialog (`make_task_library`). The dialog has not been customized.

## Customizing a Pull-Down Menu

In addition to creating custom dialogs, you can use TME 10 AEF to add and remove options from pull-down menus, including those on the main desktop. The pull-down menus on the main desktop are defined in the desktop dialog of the `AdministratorCollection` resource type.

To customize a pull-down menu:

- Retrieve the dialog descriptor.
- Reverse compile the descriptor.
- Modify the dialog’s DSL source code.
- Compile a new dialog descriptor.
- Install the new custom dialog.

As an example, we will add a `Run` menu to the managed node’s view dialog. The `Run` menu has three options: `Reboot`, `Shutdown`, and `Open xterm`. 
Customizing a Pull-Down Menu

Use the following steps to create the custom menu:

1. Use `wgetdialog` to retrieve the dialog.
   
   ```bash
   % wgetdialog -r ManagedNode view_dialog \
   > new_view.d
   ```

2. Reverse compile the dialog descriptor to obtain the dialog’s DSL source code.
   
   ```bash
   % rdsl new_view.d > new_view.dsl
   ```

   The command reverse compiles the `new_view.d` descriptor and writes the resulting DSL source code to the file `new_view.dsl`.

3. Modify the DSL source code to include the new menu item. The new lines are shown in bold in the listing below.

```plaintext
1  Command Dialog
2  {
3    Variables
4    {
5      CString dialog_name;
6      CString interpreter;
7    }
8
9    Attributes
10   {
11      BitmapTitle = $interpreter;
12      Iconic = YES;
13      Name = view_dialog;
14      Title = $dialog_name;
15    }
16
17    Gadgets
18   {
19      Collection
20     {
21        Name = managed_node_collection;
22      }
23
24      Menu
25     {
26        Attributes
```
Customizing a Pull-Down Menu

{ 
    Name = mno_system_menu;
    Title = System;
} 

Contents 
{ 
    Button 
    { 
        Commands = 
            dtc_return($owner, 
                view_dialog, 
                $self);
        HelpMessage = S_RETURN;
        Name = return;
        Title = Return;
    } 
    Button 
    { 
        Commands = 
            sync_launch%$owner();
        HelpMessage = S_CLOSE;
        Name = sync;
        Title = "Synchronize ...";
    } 
    Button 
    { 
        Commands = 
            dtc_free($owner, 
                view_dialog, 
                $self);
        HelpMessage = S_CLOSE;
        Name = close;
        Title = Close;
    } 
} 

Menu
Customizing a Pull-Down Menu

69  Attributes
70  {
71     Name = mno_edit_menu;
72     Title = Edit;
73  }
74
75  // The gadgets for this menu are
76  // not shown to conserve paper.
77  }
78
79  Menu
80  {
81     Attributes
82     {
83         Name = mno_view_menu;
84         Title = View;
85     }
86
87  // The gadgets for this menu are
88  // not shown to conserve paper.
89  }
90
91  Menu
92  {
93     Attributes
94     {
95         Name = mno_run_menu;
96         Title = Run;
97     }
98
99  Contents
100  {
101
102     Button {
103         HelpMessage = "Open an Xterm";
104         Name = xterm;
105         Title = "Xterm ...";
106         Commands = display_xterm\();
107     }
108
109     Button {
110         HelpMessage =

4. Compile a new dialog descriptor using the modified version of the dialog source code.

```
% dsl new_view.dsl > new_view.d
```

The command compiles the `new_view.dsl` DSL file and writes the dialog descriptor to `new_view.d`.

5. At this point, you can preview the dialog. The dialog has two parameter-valued variables — `dialog_name` and `interpreter` — that must be assigned values for the preview.

```
% tivoli -preview new_view.d dialog_name "'Foo'" \ interpreter "'Bar'"
```
If you select the **Run** menu on the preview dialog, you will see the new **Reboot** option.

6. Install the custom dialog descriptor using the **wputdialog** command.

   ```
   % wputdialog -r ManagedNode -T view_dialog < new_view.d
   ```

   The command installs the custom version of the **ManagedNode** resource’s **view_dialog** dialog.
When you add a field or gadget to a dialog, you must add a corresponding attribute on the instance that supports the dialog. The attribute stores the value that the user assigns to the gadget. For example, if you add a Next Maintenance field to a managed node’s dialog, you must add a corresponding attribute on the instance to store the field’s value. You must also define a callback method, which provides the interface between the dialog and the attribute. Callback methods are discussed in Chapter 7, “Custom Callback Methods.”

This chapter explains how to add attributes to resource instances. However, it does not discuss how to add properties (attributes) to an instance of a profile resource. If you want to customize an application that uses profiles, refer to Chapter 10, “TME 10 AEF and Profiles.”

### Adding a New Attribute

The `waddattr` command adds a new attribute to a specified set of resource instances. You can only add attributes that are ASCII strings. To add attributes of other types, you must use TME 10 ADE. The syntax of `waddattr` is shown below.

```
  waddattr [–v value] attribute label...
```

The `attribute` parameter is the name of the attribute to add. The `label` parameter is the object path of an instance. You can specify more than one object path to add the attribute to several instances.
Getting the Value of an Attribute

The –v option allows you to provide an initial value for the attribute. If you do not specify the –v option, the attribute is initially an empty string.

```
% waddattr -v "Building 12" location @ManagedNode:moab \n/Regions/Utah/ManagedNode:hanksville
```

The example above adds the location attribute to the moab and hanksville managed nodes. The initial value of the attribute is **Building 12**.

The waddattr command only adds the attribute to a list of instances. To add the attribute to all instances of a type, you can employ a strategy called lazy propagation. In lazy propagation, attributes are added to instances as they are needed. For example, your methods that get and set the new attribute can first check an instance for the attribute. If the attribute does not exist, the methods can use waddattr to add the attribute. See Chapter 11, “Putting It Together” for an example of lazy propagation.

Getting the Value of an Attribute

The wgetattr command returns the value of an attribute for a given resource instance. You can only use wgetattr to retrieve attributes that were added with waddattr. The command’s syntax is shown below.

```
wgetattr attribute label
```

The `attribute` parameter is the name of the attribute. The `label` parameter is the object path of the instance for which you want the attribute’s value.

The command writes the value of the specified attribute to standard output. If wgetattr must abort (if the specified attribute does not exist, for example), the command’s standard output is undefined.

```
% wgetattr location @ManagedNode:moab
Building 12
```
Setting the Value of an Attribute

The `wsetattr` command sets the value of an attribute on one or more resource instances. You can only set the values of attributes that were created with `waddattr`. The command’s syntax is shown below.

```
wssetattr value attribute label...
```

The `value` parameter is the new value for the attribute specified in the `attribute` parameter. The `label` parameter is the object path to the instance you want to change. You can specify more than one object path to set the attributes of several instances.

```
% wsetattr "Building 3" location @ManagedNode:moab
```

The example above sets the location attribute on the managed node `moab` to `Building 3`.

Removing an Attribute

The `wrmattr` command removes an attribute from a selected set of resource instances. You can only remove attributes that were created with `waddattr`. The command’s syntax is shown below.

```
wrmattr attribute label...
```

The `attribute` parameter is the name of the attribute to remove. The `label` parameter is the object path to the instance from which you want to remove the attribute. You can specify more than one object path to remove the attribute from several instances.

The example below removes the `location` attribute from an instance of the `ManagedNode` resource type.

```
% wrmattr location @ManagedNode:hanksville
```
Removing an Attribute
Custom Callback Methods

Introduction

When users request an operation from a dialog gadget, the TME 10 desktop performs an operation known as a callback method. A callback method is the means by which dialogs and gadgets respond to user input. Callbacks typically transfer data and control from the user interface to underlying application resources. For example, suppose you add a next_maint attribute to the ManagedNode resource type and add a corresponding field to the resource’s properties dialog. An important piece of this customization is creating a callback method that takes the value of the dialog gadget and sets the next_maint attribute on the resource instance to the gadget’s value.

There are several types of callback available in DSL. However, one type of callback bears special attention: legacy callback.

A legacy callback is an external script (method) that you create. Legacy callbacks can read from standard input and write to standard output, like ordinary scripts. You should always use the Bourne shell (/bin/sh) for callback methods scripts. This shell is always available in a well-known location, so TME 10 will always be able to find it to execute your scripts. Furthermore, some other shells (such as /bin/csh) make alterations to the file descriptors passed to the shell process. These alterations can cause your script to fail.

The DSL syntax for a legacy callback is shown below.

```
method @ ( args ) [< input]
```
The method is the method name. The method name must always be followed by an `@` character. The method is invoked in the context of the resource that posted the dialog. For example, if you are viewing a dialog on the moab managed node, the dialog’s callback methods are invoked on the moab resource instance.

The args parameter is an optional list of comma-separated arguments to the method, enclosed in parentheses (you must always include the parentheses, even if they are empty). The parameters are passed to the callback method as though they were specified on the command line. For example, suppose that you add this Button gadget to a dialog:

```c
Button {
    Name = digital_clock;
    Title = "Open Digital Clock";
    Commands = launch@("xclock", "-digital");
}
```

When you press the Open Digital Clock button on the dialog, the desktop invokes the launch callback method with two parameters:

- $1 = "xclock"
- $2 = "-digital"

Callback method scripts also receive environment variables passed from their invokers. These variables generally include the usual set, such as PATH, and machine-specific environment variables such as LD_LIBRARY_PATH. In addition, methods invoked as callbacks from dialogs receive the WD_DISPLAY environment variable, which contains the name of the X11 display from which the callback was invoked. For example, if your custom method is invoked from a dialog displayed on the default display on host stgeorge, the WD_DISPLAY variable passed to the method contains stgeorge:0.0.

If the method is to receive input to its standard input, you can supply it using the input parameter. You must precede the input with a angle bracket (`<`).

It is critically important that your method scripts exit with well-defined exit codes. If the script succeeded, it must exit with the exit 0 command to ensure that TME 10 understands that the script succeeded. If the script failed, it must exit with a code other than zero, else TME 10 will erroneously believe that the script succeeded.
For ease of handling, you may wish to define shell variables to represent exit codes. The three exit codes you need for TME 10 AEF scripts are shown in the table below.

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI_EXIT_OK</td>
<td>0</td>
</tr>
<tr>
<td>CLI_EXIT_ERROR</td>
<td>1</td>
</tr>
<tr>
<td>CLI_EXIT_USAGE</td>
<td>2</td>
</tr>
</tbody>
</table>

While TME 10 AEF allows you to create and install your own callback methods, it does not allow you to edit existing system callbacks. Instead, you can edit a gadget’s callback list to add pre- or post-processing steps. For example, a DSL fragment from the dialog of a fictitious sales application is shown below.

```plaintext
CommandButton {
    Name = apply;
    Title = "Apply";
    Commands = add_customer^($name, $phone, $address);
}
```

When the user presses the **Apply** button, the application invokes the `add_customer` callback method to create a new instance of the **Customer** resource. Suppose you add an attribute to the **Customer** resource; **contact_date** for example. When the user presses **Apply**, the application should also set the **contact_date** attribute. Since you can’t edit application methods, you cannot change the `add_customer` method directly. However, you can add a callback method before or after `add_customer`.

```plaintext
CommandButton {
    Name = apply;
    Title = "Apply";
    Commands = add_customer^($name, $phone, $address),
               set_contact_date@($contact_date);
}
```

With the modification, pressing the **Apply** button invokes the `add_customer` method and then invokes the `set_contact_date`, which sets the `contact_date` attribute.
Custom Method Installation

The `wputmeth` command installs a new custom method. You can use `wputmeth` to customize an instance or an entire resource type. The `wputmeth` command can install a custom method for:

- An entire resource type.
- One or more instances of a resource.
- A create dialog (defaults policy).

If there is already a custom method with the same name installed on the resource, `wputmeth` exits with an error. You must remove the existing method with `wrmmeth` before installing the new method. See “Removing a Custom Method” on page 7-14 for more information about `wrmmeth`.

Customizing an Entire Resource Type

The `wputmeth` syntax to install a custom method on a resource-wide basis is shown below.

```
wputmeth [-i interp] [-a acl]... [-u user -g group] [-t tmr...] [-T] -r resource... method path
```

The `resource` parameter is the name of the resource type you are modifying and the `method` parameter is the name of the method you are installing.

The `path` parameter to `wputmeth` describes the location of the new method you are installing. The path looks like an absolute path — it begins with a `/` character — but it is actually a relative path. TME 10 interprets the method path relative to the `/install_dir/bin` directory (where `install_dir` is the path of the TME 10 installation directory). The TME 10 `bin` directory is divided into subdirectories; one for each architecture supported in the TME 10 installation. Tivoli recommends
that you create a **CUSTOM** directory under each interpreter directory to store your custom methods.

```
/ (root)
     |
     usr
     |
     local
     |
     Tivoli
     |
     bin
     |
     man
     |
     lib
     |
     sunos4
     |
     hpx9
     |
     dgux5
     |
     CUSTOM
     |
     CUSTOM
     |
     CUSTOM
```

TME 10 resolves the method path each time a method is run, so you must leave the method script in place after you run `wputmeth`. One side effect of this is that you can change the implementation of an AEF method by replacing the script; you do not need to re-run `wputmeth`. If you want to change the location of the method script, however, you must remove the existing method with `wrmmeth` and run `wputmeth` again.

**Installing a Method for All Platform Types**

By default when you install a method, `wputmeth` assumes that the method is supported on all platform types in your installation. Use the following procedure to install a method that is supported on all available platforms:
Custom Method Installation

1. If you have not already created directories for your custom methods, create them. As you read above, Tivoli recommends that you create a directory called CUSTOM under each platform sub-directory to store your methods.

   ```bash
   % mkdir /usr/local/Tivoli/bin/aix3-r2/CUSTOM
   % mkdir /usr/local/Tivoli/bin/hpux9/CUSTOM
   ...
   % mkdir /usr/local/Tivoli/bin/uw2-ix86/CUSTOM
   ```

2. Copy your custom script into the CUSTOM directory. You can alter each copy of the script, if necessary, to account for platform-specific behavior.

   ```bash
   % cp load_xclock /usr/local/Tivoli/bin/dgux5/CUSTOM
   % cp load_xclock /usr/local/Tivoli/bin/hpux9/CUSTOM
   ...
   % cp load_xclock /usr/local/Tivoli/bin/uw2-ix86/CUSTOM
   ```

3. Run the `wputmeth` command to install the new method.

   ```bash
   wputmeth -r ManagedNode load_xclock /CUSTOM/start_xclock
   ```

   You do not need to include a platform type in the method path; TME 10 uses the value of the administrator’s `INTERP` environment variable when the method executes to locate the appropriate method script. In the case of our example method, `load_xclock`, TME 10 resolves the method path into the following absolute path when the method executes:

   `/install_dir/bin/$INTERP/CUSTOM/load_xclock`

   The `install_dir` is the TME 10 installation directory and `$INTERP` is the value of the administrator’s `INTERP` environment variable.

**Installing Platform-Specific Methods**

The default behavior of `wputmeth` assumes the method is supported on all the platforms in your installation and that the method script
exists in the same relative place under each platform binary directory. If your custom method does not fit this pattern, you must install each platform-specific version of the method separately with the \texttt{--i} option. The \texttt{--i} option on the \texttt{wputmeth} command installs a method for a single platform type. If you use the \texttt{--i} option, you must run \texttt{wputmeth} once for each platform type that you want to support your method. The possible platform types that you can use with the \texttt{--i} option are shown in the table below.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix3-r2</td>
<td>AIX 3.2.5</td>
</tr>
<tr>
<td>aix4-r1</td>
<td>AIX 4.1</td>
</tr>
<tr>
<td>dgux5</td>
<td>Data General</td>
</tr>
<tr>
<td>hpux9</td>
<td>HP-UX 9.0</td>
</tr>
<tr>
<td>hpux10</td>
<td>HP-UX 10</td>
</tr>
<tr>
<td>solaris2</td>
<td>Solaris 2</td>
</tr>
<tr>
<td>sysv4-att</td>
<td>ATT GIS (NCR)</td>
</tr>
<tr>
<td>sysv4-m88k</td>
<td>Motorola</td>
</tr>
<tr>
<td>sunos4</td>
<td>SunOS 4</td>
</tr>
<tr>
<td>uw2-ix86</td>
<td>Unixware 2.0</td>
</tr>
</tbody>
</table>

The \texttt{--i} option is actually a good choice even if your method does fit the default behavior of \texttt{wputmeth}. Installing each platform version separately gives you more flexibility than the default \texttt{wputmeth} behavior. For example, if you install your methods with the \texttt{--i} option, you can run \texttt{wrmmeth} to remove one platform’s version without affecting the method on other platforms. You cannot do this if all the platform types share the same method definition (the default with \texttt{wputmeth}).

To show an example, we will install the \texttt{load_xclock} method using the \texttt{--i} option to specify the platform type.
Custom Method Installation

1. Copy your custom script into the CUSTOM directory. You can alter each copy of the script, if necessary, to account for platform-specific behavior.

   % cp load_xclock /usr/local/Tivoli/bin/dgux5/CUSTOM
   % cp load_xclock /usr/local/Tivoli/bin/hpux9/CUSTOM
   ...
   % cp load_xclock /usr/local/Tivoli/bin/uw2-ix86/CUSTOM

2. Run the `wputmeth` command with the `–i` option to install the new method. You must run `wputmeth` once for each supported platform.

   wputmeth -i aix3-r2 -r ManagedNode load_xclock \ 
   /aix3-r2/CUSTOM/start_xclock
   ...
   wputmeth -i uw2-ix86 -r ManagedNode load_xclock \ 
   /uw2-ix86/CUSTOM/start_xclock

   Notice that the platform type is the first element of the method paths in the commands above. When you specify the `–i` option, TME 10 cannot use the administrator’s `INTERP` environment variable to complete the method path; you must explicitly include the platform type in the path.

   Once you have installed the method on all platforms, you can change the method location for a platform type (or remove it) without affecting the method on other platform types.

   % wrmmeth -i sunos4 -r ManagedNode load_xclock

Defining ACLs for a Method

Every TME 10 method has an access control list (ACL) that defines the authorization roles that are required to execute the method. When an administrator invokes a method, TME 10 checks the administrator’s roles against the method’s ACL. The TMF defines the following roles: `super`, `senior`, `admin`, `user`, and `none`. Other applications may define their own authorization roles.
When you install a custom method, you must specify the authorization roles that you want the method to support. The \(-a\) option of \texttt{wputmeth}\footnote{A command for installing custom methods.} specifies an authorization role for the method. You can use the \(-a\) option more than once to specify multiple roles. If you do not use the \(-a\) option, the method supports any authorization role.

\begin{verbatim}
% wputmeth -a super -a senior -r ManagedNode set_maint \ /CUSTOM/set_maintenance.sh
\end{verbatim}

The example above installs the \texttt{set_maint} method on the \texttt{ManagedNode} resource type. The method’s ACL is set to include the \texttt{super} and \texttt{senior} authorization roles.

**Setting the Privileges on a Method**

You can use the \(-u\) and \(-g\) options to allow the method to run with greater privileges than the invoker might have. For example, if your method must edit a system file, the method must run as \texttt{root} even if the invoker is not \texttt{root}. The \(-u\) and \(-g\) options specify the user and group ID that the method is to use when it runs.

\begin{verbatim}
% wputmeth -u root -g system -r ManagedNode get_location \ /CUSTOM/get_location.sh
\end{verbatim}

The above example installs the method \texttt{get_location} on the \texttt{ManagedNode} resource type. The method will run as the \texttt{root} user.

**Customizing a Remote TMR**

By default, the \texttt{wputmeth} command installs the method only in the TMR where the command is invoked. However, you can use the \(-t\) option to install the method in remote TMRs as well. The \(-t\) option takes a parameter, which is the name of the remote TMR.

\begin{verbatim}
% wputmeth -t Wasatch -r ManagedNode set_location \ /CUSTOM/set_location.sh
\end{verbatim}

The example above installs the \texttt{set_location} method on the \texttt{ManagedNode} resource type in the current TMR and the \texttt{Wasatch} TMR.
Custom Method Installation

You can use the –t option more than once to install the method on several remote TMRs.

% wputmeth –t France –t China –r ManagedNode set_maint \ /CUSTOM/set_maint.sh

The example above installs the set_maint method in the current TMR, the France TMR and the China TMR.

If you want to install the method in all connected TMRs, you can use the –T option instead of using the –t option to specify them all individually.

% wputmeth –T –r ManagedNode set_location \ /CUSTOM/set_location.sh

The example above installs the set_location method in the current TMR and all connected TMRs.

Customizing an Instance

TME 10 AEF allows you to install custom methods on individual instances of a resource type. To do this, use the –l option of the wputmeth command. The wputmeth syntax for installing a method on one or more instances is shown below.

wputmeth [–i interp] [–a acl]... [–u user –g group] \ –l label... method path

The syntax for customizing an instance is similar to the resource-wide customization, with the following exceptions:

- The –l option specifies the object path to an instance. You can use the –l option more than once to specify multiple instances.
- You do not have to supply the resource type with the –r option, since it should be embedded in the object path.
- The –t and –T options are irrelevant, since wputmeth automatically determines the TMR in which the instance resides.

You can find a discussion of the other wputmeth options in the section “Customizing an Entire Resource Type” on page 7-4.
An example of per-instance customization is shown below.

```
% wputmeth -l /Regions/Utah/ManagedNode:hanksville \ 
-1 @ManagedNode:moab set_location \ 
/CUSTOM/set_location.sh
```

The example installs the `set_location` method on the `hanksville` managed node and the `moab` managed node.

### Installing a Method for a Create Dialog

Dialogs used to create instances of a resource must exist independently of the instances they create. Because of this, the callback methods associated with a create dialog are stored with the resource’s defaults policies. TME 10 AEF allows you to create custom methods for create dialogs on a per-policy basis. You can use the `-c` option on the `wputmeth` command to install a custom method for a create dialog. The `wputmeth` syntax is shown below.

```
wputmeth [-i interp] [-a acl]... [-u user -g group] 
[-t tmr...] | -T] -c policy -r resource... 
method path
```

The syntax is similar to the syntax for a resource-wide customization, except that you must use the `-c` option to specify the name of a defaults policy. For a discussion of the other `wputmeth` options and parameters, see “Customizing an Entire Resource Type” on page 7-4.

An example of installing a method on a defaults policy is shown below.

```
% wputmeth -i hpx9 -c BasicManagedNode -r ManagedNode \ 
create_alias /hpx9/CUSTOM/set_location.sh
```

The example above installs an HP-UX version of the `create_alias` method on the `ManagedNode` resource’s `BasicManagedNode` defaults policy. The method can be used as a callback method for the create dialog on the `BasicManagedNode` policy.
To illustrate how to create and install a custom method, we will create a `reboot_host` method. This method is the callback of the system menu option we added in Chapter 5, “Custom Dialogs”.

A listing of the `reboot_host` method is shown below. The method shown is specific to the HP-UX platform. To be complete, you would need to create a version of the method for each supported platform.

```bash
#!/bin/sh
#
# Name: reboot_host
#
# Description: Callback method for the reboot menu option. The method takes a single
# parameter, which is a message broadcast to all users on the host.
#
# Define constants used by the method.
CLI_EXIT_OK=0
CLI_EXIT_ERROR=1
CLI_EXIT_USAGE=2
USAGE="Usage: $0 message"
MESSAGE=$1

# First check that the method was invoked correctly
if [ $# -ne 1 ]; then
  echo 2> $USAGE
  exit $CLI_EXIT_USAGE
fi

# Invoke the reboot command on the host. Use the message passed to the method as the warning message.
/etc/reboot -t 1 -m $MESSAGE
```
34 # Exit with a code indicating whether the 
35 # script succeeded or failed 
36 #
37 if [ "$?" -ne 0 ]; then 
38  exit $CLI_EXIT_ERROR 
39 fi 
40 exit $CLI_EXIT_OK 

The script runs the `/etc/reboot` command on the host in whose context 
the method is run.

Use the following steps to install the `reboot_host` method on the 
`ManagedNode` resource type.

1. If you have not already created directories for your custom 
methos, the first step is to create them. Since the `reboot_host` 
method is specific to the HP-UX platform, we will place it under 
the `hpux9` platform type.

   chmod 755 /install_dir/bin/hpux9/CUSTOM

2. Make sure that the `CUSTOM` directory has mode 755.

   chmod 755 /install_dir/bin/hpux9/CUSTOM

3. Copy the `reboot_host.sh` script to the `CUSTOM` directory. Since 
the script is specific to the HP-UX platform, it goes in the 
`hpux9/CUSTOM` directory.

   cp reboot_host.sh 
   /install_dir/bin/hpux9/CUSTOM

4. Make sure that the copy of the script has mode 755.

   chmod 755 
   /install_dir/bin/hpux9/CUSTOM/reboot_host.sh

5. Repeat steps 1–4 for every file server where TME 10 is installed.

6. Use the `wputmeth` command to install the method.

   wputmeth -i hpux9 -r ManagedNode reboot_host 
   /hpux9/CUSTOM/reboot_host.sh

Repeat the entire procedure for each platform in your installation to 
install the other platform-specific versions of the `reboot_host` method.
Removing a Custom Method

The `wrmmeth` command removes a custom method that was installed with the `wputmeth` command. You cannot use `wrmmeth` to remove existing application or framework methods.

With the `wrmmeth` command, you can remove a custom method from:
- An entire resource type.
- One or more instances of a resource type.
- A create dialog (defaults policy).

When you remove a method, `wrmmeth` does not delete the script associated with the method. You must delete the method script by hand.

Removing a Method from an Entire Resource Type

The `wrmmeth` syntax for removing a resource-wide customization is shown below.

```
wrmmeth [-i interp] [-t tmr...] [-T] -r resource... -m method
```

The `-r` option specifies the name of the resource type that contains the method you want to remove. The `method` parameter is the name of the method to remove.

Removing Platform-Specific Methods

If you are removing a method that has different versions for each platform, you must use the `-i` option to indicate which version of the method to remove. The `interp` parameter must be one of the following: `aix3-r2`, `aix4-r1`, `dgux`, `hpux9`, `hpux10`, `solaris2`, `sunos4`, `sysv4-m88k`, `sysv4-att`, or `uw2-ix86`. You must use the same platform type that was used to install the method. For example, if you installed the method with `wputmeth -i solaris2`, you must remove it using `wrmmeth -i solaris2`. If you method was installed for all platform types (no `-i` option), you cannot use the `-i` option to remove one version of the method.
Removing a Custom Method

An example of removing a platform-specific method is shown below.

% wrmmeth -i hpux9 -r ManagedNode set_location

The example above removes the HP-UX version of the set_location method on the ManagedNode resource type.

Removing Methods in Remote TMRs

By default, the wrmmeth command removes the method only in the TMR where the command is invoked. However, you can use the –t option to remove the method in remote TMRs as well. You can use the –t option more than once to remove the method in several remote TMRs.

% wrmmeth -i solaris2 -t France -t China -r ManagedNode \ set_maint

The example above removes the Solaris version of the set_maint method from the current TMR, the France TMR and the China TMR.

If you want to remove the method in all connected TMRs, you can use the –T option instead of using the –t option to specify them all individually.

% wrmmeth -i solaris2 -T -r ManagedNode set_maint

The example above removes the Solaris version of the set_maint method from the current TMR and all connected TMRs.

Removing a Method from an Instance

The wrmmeth command can remove custom methods installed on individual resource instances. You can only remove a method from an instance if it was installed as a per-instance customization on the instance. The wrmmeth syntax for removing a custom method from one or more instances is shown below.

wrmmeth [-i interp] -l label... method

The syntax for removing a custom method from an instance is similar to the resource-wide removal, with the following exceptions:
Removing a Custom Method

- The `–l` option specifies the object path to an instance. You can use the `–l` option more than once to specify multiple instances.
- You do not have to supply the resource type with the `–r` option, since it should be included in the object path.
- The `–t` and `–T` options are not used, since `wrmmeth` automatically determines the TMR in which the instance resides.

An example of removing a per-instance customization is shown below.

```bash
% wrmmeth –l /Regions/Utah/ManagedNode:hanksville
–l @ManagedNode:moab set_location
```

The example removes the `set_location` method from the `hanksville` and `moab` managed nodes. Since no interpreter type is specified, `wrmmeth` removes the method associated with the default interpreter type.

Removing a Create Dialog Method

Custom methods associated with create dialogs are stored on a resource’s defaults policies. Use the `–c` option of the `wrmmeth` command to remove a custom method from a create dialog. The `wrmmeth` syntax is shown below.

```bash
wrmmeth [-i interp] [-t tmr... | -T] –c policy
–r resource... method
```

The syntax is similar to the syntax for removing a resource-wide customization, except that you must use the `–c` option to specify the name of a defaults policy. An example of removing a method from a defaults policy is shown below.

```bash
% wrmmeth –i hpux9 –c BasicManagedNode –r ManagedNode
create_alias /hpux9/CUSTOM/set_location.sh
```

The example removes the HP-UX version of the `create_alias` method on the `ManagedNode` resource’s `BasicManagedNode` defaults policy.
Callback Method Utilities

TME 10 AEF provides several commands that allow convenient access to the TME 10 Desktop from within a callback method. For example, there are commands to

■  Display a confirmation dialog.
■  Display dialogs that contain error messages or other information.
■  Write a message to the status window on the TME 10 Desktop.

Confirmation Dialogs

It is often a good idea to confirm that the user indeed wants to perform the selected operation. TME 10 AEF provides a convenient way to generate confirmation dialogs from within a callback method. The wdispconf command displays a two-choice dialog on the desktop where the callback method is running. The command’s syntax is shown below.

```
wdispconf  –y label  –n label  message
```

The –y and –n options specify the labels of the affirmative and negative buttons, respectively. The message parameter is the text that is displayed in the dialog.

The command prints the string “YES” to standard output if the user selects the affirmative button. If the user selects the negative button, wdispconf prints “NO” to standard output.

To show how wdispconf works, consider the reboot_host method, discussed in “An Example” on page 7-12. This method can be easily modified to display a confirmation dialog before rebooting the machine.

```
1  #!/bin/sh
2  #
3  # Name: reboot_host
4  #
5  # Description: Callback method for the reboot menu
6  # option. The method takes a single
7  #    parameter, which is a message
8  #    broadcast to all users on the host.
9  #
```
Callback Method Utilities

10
11 # Define constants used by the method.
12 CLI_EXIT_OK=0
13 CLI_EXIT_ERROR=1
14 CLI_EXIT_USAGE=2
15 USAGE="Usage: $0 message"
16 MESSAGE=$1
17
18 #
19 # First check that the method was invoked correctly
20 #
21 if [ $# -ne 1 ]; then
22   echo 2> $USAGE
23   exit $CLI_EXIT_USAGE
24 fi
25
26 #
27 # Display a confirmation dialog.
28 #
29 CONFIRM = 'wdispconf -y "Yes" -n "No" \\
30   "Are you absolutely sure you want to reboot?"'
31 if [ $CONFIRM != "YES" ]; then
32   exit $CLI_EXIT_OK
33 fi
34
35 #
36 # Invoke the reboot command on the host. Use the message passed to the method as the warning message.
37 #
38 /etc/reboot -t 1 -m $MESSAGE
39
40 #
41 # Exit with a code indicating whether the script succeeded or failed
42 #
43 if [ $? -ne 0 ]; then
44   exit $CLI_EXIT_ERROR
45 fi
46
47 exit $CLI_EXIT_OK
The highlighted lines in the listing above show the part of the method that has changed. Line 30 runs the `wdispconf` command and stores the result in the `CONFIRM` variable. If the command returned something besides YES, the method exits.

Error and Message Dialogs

In addition to confirmation dialogs, TME 10 AEF provides commands to display error dialogs and message dialogs.

An error dialog is a simple dialog that contains a description of an error or system failure. The dialog has a single button, the Dismiss button, and sports the Motif error symbol.

The `wdisperr` command displays an error dialog on the desktop of the user that invokes the command. The command’s syntax is shown below.

```
wdisperr message
```

The `message` parameter is the formatted text that is to appear in the dialog.

A message dialog is a simple dialog that contains some information for the user. Like an error dialog, the message dialog has a single Dismiss button.

The `wdispmsg` dialog displays a message dialog on the desktop of the user that invokes the command. The command’s syntax is shown below.

```
wdispmsg message
```

The `message` parameter is the formatted text that is to appear in the dialog.

Other Types of Dialogs

The `wdispconf`, `wdisperr`, and `wdispmsg` commands provide a convenient way to create simple choice and message dialogs. To embed more complex dialogs in a callback method, use the
Callback Method Utilities

**wpostdialog** command. This command loads, posts, and presents a specified dialog. The command’s syntax is shown below.

```
wpostdialog [-t] dialog [{dialog-var var-value}...]
```

The *dialog* parameter specifies the name of the dialog. The dialog must exist on the same object that supports the current dialog (the dialog that invoked the callback method).

By default, **wpostdialog** makes the new dialog a child of the dialog that invoked the callback method. If you want the new dialog to be independent of the current dialog, use the `-t` option.

When you post a dialog, you must specify a value for every parameter-valued variable in the dialog. To do this, specify the variable name and its value. For example, the command below posts the **policy_results** dialog. The dialog has two parameter-valued variables, **region** and **type**.

```sh
#!/bin/sh
...
3   wpostdialog policy_results region "Southwest" \
4       type "Managed Node"
5   ...
```

If a variable is of type **CString**, and it contains blanks or other special characters, delimit the string with single quotes. For example, if the variables in the **policy_results** dialog are of type **CString**, you should invoke **wpostdialog** as shown below.

```sh
#!/bin/sh
...
3   wpostdialog policy_results region "Southwest" \
4       type "'Managed Node'"
5   ...
```

In the example above, the value of **type** is enclosed in single quotes. The value of **region** does not need single quotes, since it does not contain spaces or special characters.

If a dialog variable is used to populate a **List** or **Choice** gadget, the variable’s value must be a specially formatted string. TME 10 AEF provides a CLI command called **wformatlist**, which automatically
generates the required format from a given set of list items. The command’s syntax is shown below.

\texttt{wformatlist [list-item...]} \texttt{}

The command takes a set of list items and places them in the proper format. You must separate each item with a space. If a list item includes a space, enclose it in single quotes. For example, the method fragment below runs \texttt{wformatlist} to create a list string. The method uses the command’s result to bind the \texttt{choices} variable in the \texttt{my\_dialog} dialog.

1 #!/bin/sh
2 ...
3 LIST='wformatlist choice1 choice2 'another choice''
4 wpostdialog my_dialog choices $LIST

**Run-Time Gadget Values**

TME 10 AEF provides two commands that allow you to get and set the value of a dialog gadget from within a callback method. This can be useful for custom methods that must update status information on a dialog or must make a decision based on the state of a gadget.

The \texttt{wsetvalue} command sets the value of a dialog gadget. The command’s syntax is shown below.

\texttt{wsetvalue [–o owner –d dialog –i instance] gadget [new-value]...}

The \texttt{gadget} parameter must be the full (dot-separated) path of the gadget. See Chapter 4, “Dialog Specification Language” for a description of gadget paths.

The \texttt{new-value} parameter is new value of the gadget. For \texttt{List} and \texttt{Table} gadgets, you can specify multiple values separated by spaces.

By default, \texttt{wsetvalue} assumes the specified gadget is part of the dialog that invoked the callback method. However, you can also use \texttt{wsetvalue} to set the value of a gadget in another dialog. To do this, you must specify the following information on the command line:

- The dialog’s owner (specified with the \texttt{–o} option)
- The dialog’s name (specified with the \texttt{–d} option)
The dialog’s instance ID (specified with the –i option)

The dialog owner is the object reference of the resource that posted the dialog. The instance ID is a unique number assigned to a dialog that allows TME 10 to distinguish multiple versions of a dialog open at the same time.

The special dialog variables $owner and $self contain the owner reference and the instance ID of the current dialog. To update a parent dialog from the callback of a child dialog, you must pass $owner and $self to the callback that invokes the child. Let’s look at an example that illustrates this process.

In this example, we have a dialog that displays the status of a printer and allows you to print a file. The dialog opens a second dialog that prompts the user to enter the name of a file. A callback method of the second dialog prints the file and updates the status on the parent dialog.

A DSL listing of the parent dialog is shown below.

```
1  Command Dialog {
2    Variables {
3      Name = print_dialog;
4      Title = "Printer Status";
5      Value = "Ready for Printing";
6    }
7
8    Gadgets {
9      Message {
10         Name = printer_status
11         Value = $current_status;
12      }
13
14      CommandButton {
15         Name = print;
16         Title = "Print File...
17         Commands = select_file@$owner, $self;
18      }
19    }
20 }
```

When the user selects the Print File... button, it invokes the select_file callback. The callback takes two parameters: the object reference of the dialog’s owner and the instance ID of the dialog. As you will see,
this information eventually filters down to the callback method that prints the file, allowing the method to update the dialog’s `printer_status` gadget. The listing of the `select_file` callback method is shown below.

```
1 #!/bin/sh
2 OWNER=$1
3 DIALOG_ID=$2
4
5 wpostdialog choose_file parent_obj $OWNER \
   parent_id $DIALOG_ID
```

The `select_file` callback uses `wpostdialog` to post a dialog that prompts the user to enter the path of the file to print. The method passes the parent dialog’s information to the `choose_file` dialog as parameter-valued variables. The DSL listing of the `choose_file` dialog is shown below.

```
1 Command Dialog {
2  Variables {
3   CString parent_obj;
4   CString parent_id;
5  }
6
7  Gadgets {
8   Text {
9     Name = file_to_print;
10    Title = "File to Print";
11   }
12
13   CommandButton {
14     Name = print;
15     Title = "Print";
16     Commands = print_file@($file_to_print,
17                                $parent_obj,
18                                $parent_id);
19   }
20  }
21 }
```

When the user enters a file name and selects the `Print` button, the dialog invokes the `print_file` callback method. The dialog passes the
Callback Method Utilities

file name and the parent dialog’s information to the method. The listing of the print_file method is shown below.

```bash
#!/bin/sh
FILE=$1
STATUS_DIAG_OWNER=$2
STATUS_DIAG_ID=$3

# Some commands to start a print job omitted.

# Update the status window of the parent dialog.
wsetvalue -o $STATUS_DIAG_OWNER -d printer_status -i $STATUS_DIAG_ID print_dialog.printer_status "Now printing file: $FILE"
```

The print_file method prints the file (not shown), and then uses wsetvalue to update the printer_status gadget on the parent dialog.

The wgetvalue command returns the value of a given gadget’s Value attribute. The command’s syntax is shown below.

```bash
wgetvalue [-o owner -d dialog -i instance] gadget
```

The gadget parameter must be the full (dot-separated) path of the gadget. See Chapter 4, “Dialog Specification Language” for a description of gadget paths.

The command prints the gadget value to standard output. If the gadget has multiple values (for example, List or Table gadgets), wgetvalue prints the values one per line. An example is shown below.

```bash
#!/bin/sh

# In the code fragment above, the callback method uses wgetvalue to get the items currently selected in avail_prns list. The gadget’s path is add_printer.group1.avail_prns, which means the gadget is part of
```
the group1 gadget in the add_printer dialog. The command prints the currently selected items to standard output.

```
ps
ps-double
line
```

In this example, the items ps, ps-double, and line are currently selected in the avail_prns list.

By default, wgetvalue assumes the specified gadget is part of the dialog that invoked the callback method. However, you can also use wgetvalue to retrieve the value of a gadget in another dialog. To do this, you must specify the following information on the command line:

- The dialog’s owner (specified with the –o option)
- The dialog’s name (specified with the –d option)
- The dialog’s instance ID (specified with the –i option)

See the discussion of wsetvalue above for an example of how to obtain this information.

**Status Window**

The wstatusline command displays a message inside the status window of the TME 10 desktop. The status window is often used to provide feedback during lengthy operations. The command’s syntax is shown below.

```
wstatusline message
```

The message parameter is the formatted text of the status message.
Custom Policy

When you customize the TMP or an application, you will often need to provide policy for a new attribute or callback. Policy is the mechanism by which TME 10 administrators enforce regulations and implement strategies. This chapter discusses how to add new policy to support an AEF customization. Before attempting to create custom policy, you should be familiar with the concepts of policy and policy validation discussed in the TME 10 Framework User’s Guide.

Note: This chapter only discusses how to add and remove new custom policies. For information on how to modify existing policy methods, see the TME 10 Framework User’s Guide.

Policy is enforced through policy methods. Policy methods are standard shell scripts that do not need any special modifications to run in TME 10. Since they are standard scripts, policy methods can accept arguments, can read standard input, and can write to standard output and standard error. For example, a policy validation method takes zero or more arguments and indicates success or failure by writing “TRUE” or “FALSE” to standard output. A default policy method generates a default value and writes it to standard output.

The mechanism for enforcing policy differs between regular applications and profile-based applications. In a profile-based application, policy is defined as records within a profile. This chapter discusses how to add new policy methods to non-profile applications. Chapter 10, “TME 10 AEF and Profiles” has a full discussion of custom policy in profile-based applications.
Adding a Policy Method

When you add a new attribute to a resource, you can also provide default and validation policy for the attribute. For example, suppose you add an attribute to the managed node that identifies the disk attached to a machine. You might want to provide validation methods that control the reformatting of the disk.

The `waddpolm` command adds a new policy method to a specified resource type. The method is added to all policies defined for the resource type within the current TMR. You can also use the command to install a policy in remote TMRs. The command’s syntax is shown below.

```
waddpolm [-t tmr... | -T] [-d | -v] [-c value | -C] resource method
```

On the command line, you must specify the resource type and the name of the new policy method. A policy method can be a script or a constant value (called an intrinsic method). To create an intrinsic method, use the `-c` or `-C` option. The `-c` option creates an intrinsic method whose value is the `value` parameter. The `-C` option also creates an intrinsic method, but reads the method value from standard input. If you do not specify `-c` or `-C`, `waddpolm` assumes the method is a script and reads the script body from standard input.

The `waddpolm` command can add both new default policy methods and new validation policy methods. If you give the `-d` option on the command line, `waddpolm` adds the method to the resource type’s default policies. If you give the `-v` option, `waddpolm` adds the method to the resource type’s validation policies. If you do not specify `-d` or `-v`, `waddpolm` treats the method as a default policy method.

Note: It is important that the names of your policy methods be unique within a resource type. The names must be unique across both types of policy methods (you cannot have a default policy method and a validation policy method with the same name).

By default, `waddpolm` adds the policy method to the resource type’s policies in the TMR where the command is invoked. You can use the `-t` option to specify a list of additional TMRs where you want to install
Removing a Policy Method

the policy method. The specified TMRs must be connected to the current TMR.

% waddpolm -t France -t China -d ManagedNode \
mn_val_reboot < /work/scripts/rebootval.sh

The example above installs the mn_val_reboot default policy in the France TMR and the China TMR.

If you want to install the method in all connected TMRs, you can use the –T option instead of using the –t option to specify them all individually.

% waddpolm -T -d ManagedNode mn_val_reboot < \
/work/scripts/mn_val_reboot.sh

The example above adds the mn_val_reboot default policy to all ManagedNode policies in the current TMR and all connected TMRs.

Removing a Policy Method

The wrmpolm command removes a policy method from all policies of a resource type. You can only remove policy methods that were installed with waddpolm. The wrmpolm command’s syntax is shown below.

wrmpolm [-t tmr... | -T] [-d | -v] resource method

When you run wrmpolm, you must specify the type of resource and the name of the method to remove. The wrmpolm command can remove both default and validation policies. The –d option specifies that the method is a default policy. The –v option indicates a validation method. If you do not specify –d or –v, wrmpolm treats the method as a default policy method.

% wrmpolm -v ManagedNode mn_val_reboot

The example above removes the mn_val_reboot validation policy from the ManagedNode resource type.

By default, wrmpolm removes the policy method from the resource type’s policies in the TMR where the command is invoked. You can use the –t option to specify a list of additional TMRs where you want
to remove the policy method. The specified TMRs must be connected to the current TMR. You can also use the –T option to remove the method from all TMRs connected to the current TMR. You cannot use –t and –T at the same time. For example, to remove the mn_val_reboot validation method from the ManagedNode resource type in all TMRs, you might enter the command shown below.

% wrmpolm –T –v ManagedNode mn_val_reboot

Using Policy in a Custom Method

Policy methods are usually invoked internally from within one or more application methods. When you create a custom method or callback that must utilize policy, you can use the wrunpolm command to invoke policy methods. The command’s syntax is shown below.

wrunpolm [-i] [-p policy-region] method args...

You must specify the name of the policy method to run, and any arguments to be passed to the method. If you specify the –i option, the command’s standard input is passed to the policy method’s standard input.

If you use the –p option, wrunpolm runs the policy method on behalf of the specified region. If wrunpolm is running in the context of a method, you can omit the –p option. The command runs the policy method on behalf of the policy region where the object that invoked wrunpolm resides.

Suppose you have customized the ManagedNode resource to include an option to reboot the machine. You have also added the mn_val_reboot command to provide validation policy for the menu option. To enforce validation policy, the reboot_host method must use wrunpolm to invoke the mn_val_reboot method before executing the reboot. The reboot_host method is shown below. The policy validation is shown in bold.

```bash
#!/bin/sh
#
# Name: reboot_host
#
# Description: Callback method for the reboot menu
```
Using Policy in a Custom Method

6  # option. The method takes a single
7  # parameter, which is a message
8  # broadcast to all users on the host.
9  #
10 # Define constants used by the method.
11 CLI_EXIT_OK=0
12 CLI_EXIT_ERROR=1
13 CLI_EXIT_USAGE=2
14 USAGE="Usage: $0 message"
15 MESSAGE=$1
16
17 # # First check that the method was invoked
18 # correctly
19 #
20 if [ $# -ne 1 ]; then
21  echo $USAGE
22  exit $CLI_EXIT_USAGE
23 fi
24
25 # # Get the result of the validation method
26 #
27 RESULT='wrunpolm mn_val_reboot'
28
29 # If validation failed, exit with an error
30 if [ $RESULT != "TRUE" ]
31  exit $CLI_EXIT_ERROR
32 fi
33
34 # # Invoke the reboot command on the host. Use the
35 # message passed to the method as the warning message.
36 #
37 /etc/reboot -t 1 -m $MESSAGE
38
39 # # Exit with a code indicating whether the
40 # script succeeded or failed
41 #
42 if [ "$?" -ne 0 ]; then
43  echo "$?" "message"
44  exit $CLI_EXIT_ERROR
45 fi
46
47 #
By default, **wrunpolm** runs the policy method in the invoking object’s home policy region. If you wish to validate the object against policy in another region, use the `–p` option to specify the region’s object path. For example, the code example below checks the invoking object against the `nm_val_reboot` policy in the **Dallas** policy region.

```
wrunpolm –p /Regions/PolicyRegion:Dallas nm_val_reboot
```

There is no need to specify whether the policy method is a default or validation method. Since policy method names must be unique, TME 10 simply searches both policy types for the specified method.
Dialogs in TME 10 applications make liberal use of bitmaps, since they are an effective and efficient means of presenting information to the user. Bitmaps are often used as the title of a gadget or dialog. For example, many dialogs use buttons to transfer items to and from a list.

In addition to gadget bitmaps, such as arrows and magnifying tools, TME 10 uses bitmaps to represent instances on the desktop. These bitmaps are called icons.
Because icons represent resources, administrators can easily identify:
- The resource type of an instance.
- The state of an instance.

TME 10 AEF allows you to create and edit custom bitmaps and icons using the bitmap editors available on your system.

XPM Bitmap Format

The TME 10 Desktop uses bitmaps and icons that are stored in the X pixmap (XPM) format. The XPM format is used extensively in X Windows, since it allows an image to be displayed on monochrome, grey scale, or color displays.

Many icon editors support the XPM format (vueicon on HP-UX, for example). If your system does not have an XPM editor, a public domain pixmap editor is included in the Tivoli Productivity Package. This package is a collection of tools that help you get the most out of TME 10. A manual page for the editor is included on the Productivity CD-ROM. See the Introduction to TME 10 ADE for information about the Productivity Package.

If you want to use your favorite bitmap editor and it does not support XPM, you can still create TME 10 bitmaps and icons. Use the procedure below to convert your bitmaps into the format required by TME 10.

1. Create a color version of the bitmap.
2. Create a black and white version of the bitmap.
3. Create a mask (reverse) of the bitmap. The mask is only used with icon bitmaps.
4. Convert the bitmap to portable pixmap (PPM) format.
5. Convert the black and white, and mask versions into X11 bitmap (XBM) format.
6. Use the wconbitmap command to combine the PPM and XBM bitmaps into a single XPM bitmap.

To convert bitmaps into PPM and XBM formats, TME 10 includes a public domain package of filters called PBMPLUS.
created by Jef Poskanzer, can convert many common bitmap formats, including:

- GIF
- PICT
- PCX
- TrueVision Targa
- HP PaintJet
- Sun icon
- X10 and X11 bitmaps (XBM)
- MacPaint
- GEM
- And many others

The PBMPLUS filters are part of the Tivoli Productivity Package, which is a collection of tools that help you get the most out of TME 10. See the TME 10 Framework User’s Guide for more information.

After you create the bitmap versions and convert them into PPM and XBM formats, you must use the `wconbitmap` command to combine the bitmaps into a single XPM bitmap. The command’s syntax is shown below.

```
wconbitmap ppm ppm_color xbm_bw xbm_mask > xpmfile
```

The `ppm_color` parameter is the file name of the color version of the bitmap. The `xbm_bw` parameter is the file name of the black and white version of the bitmap. The `xbm_mask` parameter is the file name of the bitmap mask.

The `wconbitmap` command converts the three files into a single XPM file. The command writes the XPM file to standard output; you can redirect this output to a file.

To give an example of how to convert a bitmap, suppose you want to create your bitmaps with a GIF editor. In this case, you would use the procedure below to create the XPM bitmap.
1. Use the GIF editor to create three versions of the bitmap: color, black and white, and mask.

2. Convert the color version of the bitmap from GIF to PPM format.

   \% giftoppm foo.color.gif > foo.color.ppm

3. Convert the black and white version from GIF to XBM format.

   \% giftoppm foo.bw.gif | ppmtopgm | pgmtopbm | pbmtoxbm > foo.bw.xbm

4. Convert the mask from GIF to XBM.

   \% giftoppm foo.mask.gif | ppmtopgm | pgmtopbm | pbmtoxbm > foo.mask.xbm

5. Use the \texttt{wconbitmap} command to convert the color, black and white, and mask versions into a single XPM format file.

   \% wconbitmap ppm foo.color.ppm foo.bw.xbm \ foo.mask.xbm > foo.xpm

The next section discusses how to install your custom bitmaps.

**Bitmaps**

Dialogs in TME 10 applications make liberal use of bitmaps, since they are an effective and efficient means of presenting information to the user. Bitmaps are often used as the title of a gadget or dialog. For example, many dialogs use buttons to transfer items to and from a list.

The \texttt{BitmapTitle} attributes of the button gadgets in the picture above are set to the names of the \texttt{[E]} and \texttt{[W]} arrow bitmaps. See Chapter 4, “Dialog Specification Language” for information about the \texttt{BitmapTitle} attribute.
You can use TME 10 AEF to add new bitmaps to a resource or to customize an existing bitmap. To understand how bitmaps and customizations are organized in TME 10, look at the figure below.

Like dialogs, bitmaps are stored in a resource’s user interface component. All instances of the resource use these bitmaps. There are also two extension areas: one on the resource and one on each instance. This allows you to install custom bitmaps that affect all instances of a resource or individual instances.

The bitmaps for a “create” dialog are stored on the resource’s defaults policies, which is also where the create dialogs are stored. Each defaults policy has an extension component that allows you to customize the create dialog. You can create custom dialogs that affect all defaults policies or individual policies.
Getting a Bitmap

You can use the `wgetbitmap` command to retrieve:

- Bitmaps stored in a resource’s user interface and extension components.
- Custom bitmaps stored in an instance’s extension component.
- Bitmaps stored in a policy resource’s user interface and extension components.
- Custom bitmaps stored in an individual policy’s extension component.

Getting a Bitmap Stored on a Resource

The `wgetbitmap` syntax for retrieving a bitmap stored on a resource is shown below.

```
wgetbitmap -r resource [-t tmr] bitmap-name > bitmap
```

The `-r` option specifies the resource type that uses the bitmap. The `bitmap-name` parameter is the name of the bitmap. If there is a custom version of the bitmap, `wgetbitmap` returns it. Otherwise, the command returns the original version of the bitmap.

The command writes the bitmap to standard output. You can redirect it to a file. An example is shown below.

```
% wgetbitmap -r ManagedNode sunos4-client > sunos.xpm
```

The example retrieves the `sunos4-client` bitmap from `ManagedNode` resource and redirects the output to a file called `sunos.xpm`.

By default, the `wgetbitmap` command works on the given resource in the TMR where the command is invoked. However, you can use the `-t` option to get a bitmap from a remote TMR. The `-t` option takes a parameter, which is the name of a TMR that is connected to the current TMR.

```
% wgetbitmap -t Uinta -r ManagedNode sunos4-client > sunos.xpm
```

The example gets the `sunos4-client` bitmap from the `ManagedNode` resource in the TMR `Uinta`.
Getting a Per-Instance Custom Bitmap

The `wgetbitmap` command can retrieve custom bitmaps that have been installed directly on an instance of a resource. The `wgetbitmap` syntax for getting a per-instance custom bitmap is shown below.

```
wgetbitmap -l label bitmap-name > bitmap
```

The `-l` option specifies the label path to an instance. You do not have to use the `-r` option to specify the resource type, since it must be included in the label path. You can use the name registry form of the label path (`@type:label`) to get a bitmap from an instance in a remote TMR. This is because the name registry in the current TMR is automatically aware of instances that exist in other TMRs.

If there is no custom version of the bitmap on the instance, `wgetbitmap` exits with an error. If there is a custom bitmap, the command writes it to standard output. You can output the bitmap to a file. An example of per-instance customization is shown below.

```
% wgetbitmap -l @ManagedNode:moab sunos4-client > sunos.xpm
```

The example retrieves the `sunos4-client` bitmap from the managed node `moab` and redirects the output to a file called `sunos.xpm`.

Getting a Bitmap From a Create Dialog

You can use `wgetbitmap` to retrieve a bitmap that is used by a create dialog. As you read in the introduction of this chapter, each defaults policy of a resource contains a create dialog. Bitmaps used by a create dialog are stored on the defaults policy with the dialog. Just as bitmaps can be customized on a resource-wide basis or per-instance basis, bitmaps can be customized for an entire policy resource or for a specific policy.

Use the syntax below to retrieve a bitmap that is stored on a policy resource (i.e., available to all create dialogs of a type).

```
wgetbitmap -C -r resource [-t tmr] bitmap-name > bitmap
```
If the bitmap has been customized for the entire resource type, `wgetbitmap` returns the custom version. Otherwise, the command returns the original version. An example is shown below.

```
% wgetbitmap -C -r TaskLibrary up_arrow > custom_up.xpm
```

In the example, `wgetbitmap` checks for a custom version of the `up_arrow` bitmap. If there is no custom version, `wgetbitmap` returns the original version. The bitmap is redirected to the file `custom_up.xpm`.

Custom bitmaps can also be installed on individual policies. Use the syntax shown below to retrieve a per-policy customization.

```
wgetbitmap -c policy -r resource [-t tmr] bitmap-name
```

The `-c` option causes `wgetbitmap` to look for a custom bitmap stored on the policy specified by the `policy` parameter. If there is a per-policy customization, `wgetbitmap` returns it. If there is no per-policy customization, the command checks for a custom version for the entire policy resource. If there is no resource-wide custom version, `wgetbitmap` returns the original bitmap. An example is shown below.

```
% wgetbitmap -c BasicTaskLibrary -r TaskLibrary up_arrow > custom_up.xpm
```

The example looks for a custom bitmap, `up_arrow`, on the `BasicTaskLibrary` defaults policy. If there is no custom version for this policy, `wgetbitmap` checks for a customization that applies to all create dialogs. If there is no customization across all defaults policies, the command returns the original version of `up_arrow`.

As with resource-wide customizations, the `wgetbitmap` command normally works with defaults policies in the TMR where the command is run. You can use the `-t` option to get a bitmap from a remote TMR.

### Installing a Custom Bitmap

The `wputbitmap` command installs a custom bitmap. You can use `wputbitmap` to install customizations of existing bitmaps or to add your own original bitmaps. When you install a customization for an
existing bitmap, the original is not destroyed. The customization simply overrides the original.

The \texttt{wputbitmap} command allows you to customize bitmaps for:

- An entire resource type.
- Specific instances of a resource type.
- A create dialog.

If the bitmap is associated with a create dialog, you can install the bitmap on all defaults policies or on individual policies.

\textbf{Installing a Resource-Wide Bitmap}

The \texttt{wputbitmap} syntax for installing a resource-wide customization is shown below.

\begin{verbatim}
\texttt{wputbitmap \textendash r resource \textendash t tmr... | \textendash T} \texttt{bitmap-name < bitmap}
\end{verbatim}

The \texttt{\textendash r} option specifies the resource type that uses the bitmap. The \texttt{bitmap-name} parameter is the name of the bitmap. The name is used to reference the bitmap from within TME 10.

The \texttt{wputbitmap} command reads the bitmap from standard input. You can redirect the XPM file to the command’s standard input, as shown in the example below.

\begin{verbatim}
% wputbitmap \textendash r TaskLibrary left_arrow < custom_left.xpm
\end{verbatim}

The example installs a resource-wide customization of the \texttt{left_arrow} bitmap for the \texttt{TaskLibrary} resource type. The bitmap definition is redirected from a file called \texttt{custom_left.xpm}.

By default, the \texttt{wputbitmap} command installs the custom bitmap on the resource in the TMR where the command is invoked. However, you can use the \texttt{\textendash t} option to install the bitmap in remote TMRs. The \texttt{\textendash t} option takes a parameter, which is the name of a TMR that is connected to the current TMR. You can specify the \texttt{\textendash t} option more than once to install the bitmap in multiple TMRs.

\begin{verbatim}
% wputbitmap \textendash t Wasatch \textendash t LaSalle \textendash r TaskLibrary \textbf{\textendash t LaSalle} \textendash r TaskLibrary \textbf{\textendash t LaSalle} < custom_left.xpm
\end{verbatim}
The example installs a resource-wide customization of the `left_arrow` bitmap for the `TaskLibrary` resource type. The bitmap is installed in the `Wasatch` and `LaSalle` TMRs.

If you want to install the bitmap in all connected TMRs, you can use the `–T` option instead of using the `–t` option to specify them all individually. When you use the `–T` option, `wputbitmap` also installs the bitmap in current TMR.

```
% wputbitmap –r TaskLibrary –T left_arrow < custom_left.xpm
```

The example above installs the `left_arrow` bitmap on the `TaskLibrary` resource in the current TMR and all connected TMRs.

### Installing a Bitmap On an Instance

The `wputbitmap` syntax for installing a custom bitmap on an instance is shown below.

```
wputbitmap –l label bitmap-name < bitmap
```

The `–l` option specifies the label path to an instance. You do not have to use the `–r` option to specify the resource type, since it must be included in the label path.

You can use the name registry form of the label path (`@type:label`) to install the bitmap on instances in remote TMRs. This is because the name registry in the current TMR is automatically aware of instances that exist in other TMRs.

The `wputbitmap` command reads the bitmap from standard input. You can redirect the XPM file on the command line. An example of per-instance customization is shown below.

```
% wputbitmap –l @ManagedNode:moab up_arrow < custom_up.xpm
```

The example installs an `up_arrow` bitmap on the managed node `moab`. The bitmap file is redirected from the file `custom_up.xpm`.

### Installing a Bitmap For a Create Dialog

You can use `wputbitmap` command to install a custom bitmap on a create dialog. As you read in the introduction of this chapter, each default policy of a resource contains a create dialog. Bitmaps used by
a create dialog are stored on the defaults policy with the dialog. The syntax for installing a create dialog bitmap depends on whether you want to customize the bitmap on all defaults policies or on a specific policy.

If you want to install the bitmap on all defaults policies, use the syntax shown below.

```
 wputbitmap {–C | –c policy} –r resource {–t tmr... | –T} bitmap-name < bitmap
```

The –C option installs a create dialog on all defaults policies of a resource type. An example is shown below.

```
% wputbitmap –C –r TaskLibrary left_arrow < custom_left.xpm
```

The example installs a custom bitmap, `left_arrow`, on all defaults policies of the `TaskLibrary` resource. The body of the bitmap is redirected from the file `custom_left.xpm`.

The –c option installs a bitmap on a specific defaults policy. An example is shown below.

```
% wputbitmap –c BasicTaskLibrary –r TaskLibrary \ 
   up_arrow < custom_up.xpm
```

The example installs a custom bitmap, `up_arrow`, on the `BasicTaskLibrary` defaults policy. The body of the bitmap is redirected from the file `custom_up.xpm`.

As with resource-wide customizations, the `wputbitmap` command normally works with defaults policies in the TMR where the command is run. You can use the –t and –T options to install bitmaps on defaults policies in remote TMRs.

### Removing a Custom Bitmap

The `wrmbitmap` command removes a custom bitmap from an instance or resource type. You can only remove bitmaps that were installed with `wputbitmap`. The `wrmbitmap` command allows you to remove custom bitmaps from:

- An entire resource type.
- Specific instances of a resource type.
Create dialogs.

If the bitmap is associated with a create dialog, you can remove the bitmap from all defaults policies or from individual policies.

You can only remove a bitmap in the same context as it was installed. For example, if you installed the bitmap as a resource-wide customization, you must remove it on that basis. You cannot install a resource-wide bitmap and then remove it from a few instances.

Removing a Resource-Wide Bitmap

The `wrmbitmap` syntax for removing a resource-wide bitmap customization is shown below.

```
wrmbitmap -r resource [-t tmr...] [-T] bitmap-name
```

The `-r` option specifies the resource type that uses the bitmap. The `bitmap-name` parameter is the name of the bitmap to remove.

An example of `wrmbitmap` is shown below.

```
% wrmbitmap -r TaskLibrary left_arrow
```

The example removes the resource-wide customization of the `left_arrow` bitmap from the `TaskLibrary` resource.

By default, the `wrmbitmap` command works in the TMR where the command is invoked. However, you can use the `-t` option to remove a bitmap in remote TMRs. The `-t` option takes a parameter, which is the name of the remote TMR. You can specify the `-t` option more than once to remove the dialog in multiple TMRs.

```
% wrmbitmap -r TaskLibrary -t "Paris" left_arrow
```

The example above removes the `left_arrow` bitmap from the `TaskLibrary` resource in the `Paris` TMR.

If you want to remove the customization in all connected TMRs, you can use the `-T` option instead of using the `-t` option to specify each TMR individually. When you use the `-T` option, `wrmbitmap` also removes the dialog in current TMR.

```
% wrmbitmap -r TaskLibrary -T left_arrow
```
Bitmaps and Icons

The example above removes the left_arrow custom bitmap from the TaskLibrary resource in the current TMR and all connected TMRs.

Removing a Bitmap From an Instance

The wrmbitmap command can remove a bitmap that was installed as a per-instance customization. The syntax for removing a per-instance customization is shown below.

```
wrmbitmap -l label bitmap-name
```

The –l option specifies the label path to an instance. You do not have to use the –r option to specify the resource type, since it must be included in the label path. The bitmap-name parameter is the name of the bitmap to remove.

```
% wrmbitmap -l @TaskLibrary:tools smiley
```

The example above removes the per-instance customization of the smiley bitmap from the task library tools.

Removing a Bitmap From a Create Dialog

You can use wrmbitmap to remove a custom bitmap from a create dialog. As you saw in the section on installing bitmaps, the bitmaps associated with a create dialog are stored with a resource’s defaults policies. The syntax for removing a create dialog bitmap is shown below.

```
wrmbitmap -C -r resource [-t tmr... | -T] bitmap-name
```

The –C option removes the bitmap from all the defaults policies of a resource type. An example is shown below.

```
% wrmbitmap -C -r TaskLibrary up_arrow
```

In the example, wrmbitmap removes the up_arrow bitmap from every default policy defined for the TaskLibrary resource.
Bitmaps

Note: The `wputbitmap` command allows you to install a custom bitmap on an individual defaults policy or on all defaults policies associated with a given resource type. However, the `wrmbitmap` command only supports resource-wide removal of custom bitmaps installed on defaults policies. Future versions of the command will support the ability to remove custom bitmaps from individual defaults policies.

As with resource-wide customizations, the `wrmbitmap` command normally works with defaults policies in the TMR where the command is run. You can use the `–t` and `–T` options to remove bitmaps from defaults policies in remote TMRs.

Getting a List of Bitmaps for a Resource

The `wlsbitmap` command provides a list of bitmaps on a resource type. The command also lists the bitmaps for which there are resource-wide customizations. It does not show bitmaps that have been customized on individual instances. The syntax of `wlsbitmap` is shown below.

```
wlsbitmap [-C] -r resource [-t tmr... | -T]
```

The `–r` option specifies the type of resource. The `–t` and `–T` option can be used to get a list of dialog from one or more remote TMRs.

The output of `wlsbitmap` depends on which language environment (locale) you are using. The output generated in the C locale has the following format:

```
bitmap1 (customization status)
bitmap2 (customization status)
...
```

The command lists each bitmap name. If there is a custom version of the bitmap, `wlsbitmap` notes it. An example of `wls bitmap` is shown below.

```
% wlsbitmap -r TaskLibrary
bitmap name (customization status)
  task_library
  task (resource-wide customization)
  job
```

```
```
The example lists the bitmaps for the TaskLibrary resource. The task bitmap is a resource-wide customization.

The wlsbitmap command’s --C option provides information about the bitmaps used on a resource’s create dialog. This option shows any custom bitmaps installed across all defaults policies. It does not show custom bitmaps installed on individual policies. An example is shown below.

```
% wlsdialog --C -r TaskLibrary
bitmap name (customization status)
task_library
```

The example shows that the TaskLibrary resource has a single bitmap for its create dialog. The bitmap has not been customized.

**Icons**

The TME represents instances of resources as icons on the TME Desktop. Since each resource type has its own unique icon, it is very easy to determine an instance’s type just by looking at its icon.

Resource types actually can have several icons. Which icon TME 10 uses to represent an instance of a resource depends on the state of the instance. A resource type can have one or more states, each with its own icon. For example, an administrator’s bulletin board can be in one of two states. In one state, there are no unread notices on the bulletin board.
Icons

board. In other state, there are notices that the administrator has not yet read.

As you can see from the figure above, the bulletin board moves back and forth between the two states. When there are no unread notices, the board is in the `no_unread` state, and the icon appears as an empty bulletin board.

When an application posts a notice to the bulletin board, the board changes states from `no_unread` to `unread`. Consequently, the icon now appears brimming with notices.
With TME 10 AEF, you can create new states for a resource and assign a custom icon to the state. Then you can set the states of individual instances. This causes the instance to be represented on the TME 10 Desktop by your custom icon.

Note: You can change an instance’s state, but you cannot change an application so that it recognizes your custom state. If the application changes the state of an instance that you have customized, you must reset the state by hand.

The icon associated with each state is composed of a bitmap and a dialog descriptor. The bitmap is the visual representation of the icon.

The dialog descriptor provides a DSL definition for an icon gadget and its pop-up menu. An example of an icon menu definition is shown below.

```
1 Command Dialog
2 {
3
4 //
5 // Icon Gadget
6 //
7
8     Icon {
9         Name = icon;
10         Type = ManagedNode;
11         Commands = icon_menu;
12     }
13
14 //
15 // Icon Menu
16 //
17
18     Menu {
19         Name = icon_menu;
20         Title = "Managed Node";
```
The first part of the DSL listing (lines 8–12) defines an Icon gadget for the icon. The Icon gadget provides a communication link between the icon and the Desktop. Line 9 names the icon and line 10 sets the icon’s Type attribute. Icons are categorized into types so that the Desktop knows what kinds of drag-and-drop operations the icon can perform. The icon’s Commands attribute (line 11) is set so that the icon_menu gadget displays when the icon is selected.

The icon_menu gadget, defined on lines 18–39, is a Menu gadget that contains three buttons (choices):

- **Open...**
- **Managed Node Properties**
- **Run xterm**

The Open... choice invokes the display_view callback method. The Managed Node Properties choice invokes the show_properties callback method. The Run xterm choice invokes the display_xterm callback method.
Use the following steps to create a custom icon:

1. If you want to modify an existing bitmap, use `wgetbitmap` to get it. Otherwise, create a new XPM icon bitmap from scratch.

2. Create a new icon dialog or use `wgetdialog` to get a copy of an existing dialog descriptor.

3. Use the `wputicon` command to define a new state on the resource and install your custom icon.

4. Use the `wputstate` command to change selected instances so they use your new state.

If you want to change a custom icon that you have already created, use the following steps:

1. Use `wgetbitmap` or `wgetdialog` to get the state’s bitmap or dialog.

2. Add your customization.

3. Use `wputbitmap` or `wputdialog` to install the new version of the bitmap or dialog.

## Getting Icon Dialogs and Bitmaps

The first step to customizing an icon is to retrieve information about the original version of the icon. To do this, you must

- Use `wgetdialog` to retrieve the icon’s dialog descriptor.
- Use `wgetbitmap` to retrieve the icon’s XPM bitmap.

The syntax for getting an icon’s dialog and bitmap are shown below.

```
wgetdialog -I state -r resource [-t tmr] > dialog.d
wgetbitmap -I state -r resource [-t tmr] > bitmap.xpm
```

The `-I` option instructs the commands to return the dialog or bitmap for the state named by the `state` parameter. The `-r` option specifies the resource type for which the state is defined.

You can use the `-t` option to get an icon from a remote TMR. The TMR must be connected to the TMR where the command is run.
Icons

The `wgetdialog` command writes the icon’s dialog descriptor to standard output. You can then use the `rdsl` compiler to convert the descriptor into DSL source code. See Chapter 5, “Custom Dialogs” for more information about the `rdsl` compiler.

The `wgetbitmap` command writes the icon’s bitmap to standard output. The bitmap is stored in XPM format. For more information about bitmap formats, see “XPM Bitmap Format” on page 9-2.

To show an example, the commands below get the dialog and bitmap for the `hpux9-client` state of the `ManagedNode` resource type.

```
% wgetdialog –I hpux9-client –r ManagedNode > HP_icon_dlg.d
% wgetbitmap –I hpux9-client –r ManagedNode > HP_icon.xpm
```

The first command above redirects the dialog descriptor to the file `HP_icon_dlg.d`. The second command redirects the icon’s bitmap to the file `HP_icon.xpm`.

**Customizing an Icon**

After you retrieve the icon’s dialog and bitmap, you can customize either to create a new icon.

To customize the icon’s bitmap, you can use the public domain pixmap editor that comes on the Tivoli Productivity Package CD-ROM. You can also create a new icon in another format (GIF or JPG, for example) and convert it to XPM. See “XPM Bitmap Format” on page 9-2 for more information about creating new bitmaps.

To customize the icon’s dialog, you must:

1. Reverse compile the descriptor to get the DSL source.
2. Modify the DSL source to add your customization.
3. Use the `dsl` compiler to create a new dialog descriptor.

Chapter 5, “Custom Dialogs” contains a discussion of how to customize a dialog. If you are new to DSL, see Chapter 4, “Dialog Specification Language” for an introduction.
Icons

Installing an Icon

With TME 10 AEF, you can create a new state for a resource and install an icon for that state. You can also use the `wputbitmap` and `wputdialog` commands to replace bitmaps and icon dialogs for states that were created with TME 10 AEF.

Creating a New Icon State

The `wputicon` command defines a new state for a resource and associates your icon with the new state. The command’s syntax is shown below.

```
wputicon -r resource [-t tmr... | -T] state dialog-file bitmap-file
```

The `–r` option specifies the resource on which you are defining the new state. The `state` parameter is the name of the new state. If the resource already has a state with the given name, `wputicon` exits with an error.

The `dialog-file` parameter is the path to the icon’s dialog descriptor, and `bitmap-file` is the path to the icon’s bitmap. The command makes copies of these files, so you can remove them after you install the icon.

```
% wputicon -r ManagedNode internet_gate gateway.d \
  gateway.xpm
```

The example above defines a new state called `internet_gate` on the `ManagedNode` resource type. The new state’s icon is defined by the `gateway.d` dialog descriptor and the `gateway.xpm` bitmap.

As with other TME 10 AEF commands, `wputicon` defines the new state in the TMR where the command is invoked. You can use the `–t` option to install the icon in a remote TMR. You can use `–t` more than once to install the icon in multiple TMRs.

```
% wputicon -r ManagedNode internet_gate -t Wasatch \
  -t LaSal gateway.d gateway.xpm
```

The example above defines the `internet_gate` state on the `ManagedNode` resource in the `Wasatch` and `LaSal` TMRs.

If you want to install the dialog in all connected TMRs, you can use the `–T` option instead of using the `–t` option to specify them all.
Icons

individually. When you use the –T option, **wputicon** also installs the dialog in current TMR. An example is shown below.

% wputicon -r ManagedNode internet_gate -T \  
gateway.d gateway.xpm

The example defines the **internet_gate** state on the **ManagedNode** resource in the current TMR and all connected TMRs.

**Replacing an Icon Bitmap**

The **wputbitmap** command can be used to install a custom bitmap on an existing state. You can only replace the bitmap for a state that was created with **wputicon**. The syntax of **wputbitmap** is shown below.

**wputbitmap –I** state **–r** resource [[-t tmr]... | -T] < bitmap.xpm

The –I option causes **wputbitmap** to replace the bitmap for the state named by the state parameter. The –r option specifies the resource type for which the state is defined.

The **wputbitmap** command reads the bitmap from standard input, so you must redirect the XPM file on the command line.

% wputbitmap –I internet_gate –r ManagedNode  
- gateway.xpm

The example above replaces the **internet_gate** state’s icon bitmap with the bitmap in the file **new_gateway.xpm**.

As with other TME 10 AEF commands, the **wputbitmap** command normally works in the TMR where the command is run. You can use the –t and –T options to install custom icon dialogs in remote TMRs.

**Replacing an Icon Dialog**

The **wputdialog** command can be used to install a custom icon dialog on an existing state. You can only replace the dialog if the state was created with **wputicon**. The syntax of **wputdialog** is shown below.

**wputdialog –I** state **–r** resource [-t tmr... | -T] < dialog.d
Icons

The –I option causes \texttt{wputdialog} to replace the icon dialog for the state named by the \textit{state} parameter. The –r option specifies the resource type for which the state is defined.

The \texttt{wputdialog} command reads the dialog descriptor from standard input, so you must redirect the descriptor file on the command line.

\begin{verbatim}
% wputdialog –I internet_gate –r ManagedNode < new_gateway.d
\end{verbatim}

The example above replaces the \texttt{internet_gate} state’s dialog descriptor with the descriptor in the file \texttt{new_gateway.d}.

As with other TME 10 AEF commands, the \texttt{wputdialog} command normally works in the TMR where the command is run. You can use the –t and –T options to install custom icon dialogs in remote TMRs.

\section*{Using the New Icon}

Installing a custom icon does not cause instances of the resource to begin using it. You must change the state of specific instances. The \texttt{wputstate} command sets the state of an instance. When you change the state of an instance, the TME 10 Desktop begins using the icon that is associated with the state. The \texttt{wputstate} command’s syntax is shown below.

\begin{verbatim}
wputstate –l label... state
\end{verbatim}

The \texttt{label} parameter is the object path of the instance whose state you want to change. You must include the resource type in the object path. You can use the name registry form of the label path (@\texttt{type}:\texttt{label}) to change the state of an instance in a remote TMR. This is because the name registry in the current TMR is automatically aware of instances in other, connected TMRs.

The \texttt{state} parameter is the name of the instance’s new state. If you specify an invalid state name, \texttt{wputstate} exits with an error. An example of \texttt{wputstate} is shown below.

\begin{verbatim}
% wputstate @ManagedNode:moab internet_gate
\end{verbatim}
The example above sets the managed node `moab` to the state `internet_gate`. The TME 10 Desktop will now use your custom icon to represent `moab`.

If you need to determine the current state of an instance, use the `wgetstate` command. The command’s syntax is shown below.

```
wgetstate -l label
```

The command writes the state of the instance to standard output. An example is shown below.

```
% wgetstate -l @ManagedNode:moab
internet_gate
```

In the example, the current state of the managed node `moab` is `internet_gate`.

**Examples**

To further illustrate how to create a custom icon, this section presents two full examples. In the first example, we will define a new icon state for the `ManagedNode` resource type. In the second example, we will add a new option to our icon’s pop-up menu.

**Customizing the Icon Bitmap**

For this example, we will create a new a icon for the `ManagedNode` resource type. The new icon, shown below, will represent managed nodes that have internet connections to the outside world.

1. Use the `wgetdialog` command to retrieve the icon menu from an existing managed node state. It doesn’t matter which state you get it from, since all managed node states use the same icon menu.

```
% wgetdialog -I sunos4-client -r ManagedNode \\
> icon_menu.d
```
2. Since you are creating an icon bitmap from scratch, you do not need to get an existing bitmap with `wgetbitmap`. Instead, create the new globe icon using a bitmap editor. The editor in the picture below is the `pixmap` editor that comes with TME 10.

![Bitmap Editor](image)

3. Use the `wputicon` command to define a new state and install your custom icon. The new state is called `internet_gate`.

```
% wputicon -r ManagedNode internet_gate icon_menu.d \
globe.xpm
```
4. Use the `wputstate` command to set the state of any managed nodes that are Internet gateways.

```
% wputstate @ManagedNode:moab @ManagedNode:saltlake \ internet_gate
```

The managed nodes **moab** and **saltlake** now appear on the TME 10 Desktop like so:

![Icon Menu Screenshot]

**Customizing the Icon Menu**

Now that you have created a new icon state, we will add a `reboot` command to its pop-up menu. This example uses the `reboot_node` callback method discussed in Chapter 7, “Custom Callback Methods.”

1. Use the `wgetdialog` command to retrieve the icon menu for the `internet_gate` state.

```
% wgetdialog -I internet_gate -r ManagedNode > gate.d
```
2. Reverse compile the dialog descriptor to obtain the dialog’s DSL source code.

    % rdsl gate.d > new_gate.dsl

The command reverse compiles the `gate.d` descriptor and writes the resulting DSL source code to the file `new_gate.dsl`.

3. Modify the DSL source code to include the new menu item. The new lines are shown in bold in the listing below.

```plaintext
Command Dialog
{
   //
   // Icon Gadget
   //
   Icon {
      Name = icon;
      Type = ManagedNode;
      Commands = icon_menu;
   }

   //
   // Icon Menu
   //
   Menu {
      Name = icon_menu;
      Title = "Managed Node";
      Button {
         Name = open_tool;
         Title = "Open ..."
         Commands = display_view%();
      }
      Button {
         Name = run_properties;
         Title = "Managed Node Properties"
         Commands = show_properties%();
      }
   }
```
Icons

33           Button {
34               Name = reboot;
35               Title = "Reboot...";
36               Commands = reboot_host@();
37           }
38
39           Button {
40               Name = xterm;
41               Title = "Run xterm";
42               Commands = display_xterm%;
43           }
44       }
45

4. Compile a new dialog descriptor using the modified version of the DSL source code.

   % dsl new_gate.dsl > new_gate.d

   The command compiles the new_gate.dsl DSL file and writes the dialog descriptor to new_gate.d.

5. Install the custom dialog descriptor using the wputdialog command.

   % wputdialog –I internet_gate –r ManagedNode < new_gateway.d

Removing a Custom Icon

The wrmicon command removes a custom icon that was installed with wputicon. The command’s syntax is shown below.

   wrmicon –r resource state [–t tmr... | –T]

   The –r option specifies the resource from which you are removing the state. The state parameter is the name of the state. If the given state was not created with wputicon, wrmicon exits with an error.

   % wrmicon –r ManagedNode internet_gate

   The example above removes the internet_gate state from the ManagedNode resource type.
As with other TME 10 AEF commands, \texttt{wrnicon} works within the current TMR by default. You can use the \texttt{-t} and \texttt{-T} options to remove the icon from remote TMRs.

## Getting Lists of Custom Icons

The \texttt{wlsicon} command lists all the icon states defined for a resource type. The command’s syntax is shown below.

\texttt{wlsicon \texttt{-r} resource [\texttt{-t} tmr]}

The \texttt{-r} option specifies the resource type. You can use the \texttt{-t} option to get the states of a resource in a remote TMR.

A sample of the command’s output is shown below.

```
% wlsicon \texttt{-r} ManagedNode \texttt{-t} Wasatch
sunos4-client
sunos4-server
solaris2-client
solaris2-server
hpux9-client
hpux9-server
sysv4-m88k-client
sysv4-m88k-server
aix3-r2-client
aix3-r2-server
dgux5-client
dgux5-server
sysv4-att-client
sysv4-att-server
uw2-ix86-client
uw2-ix86-server
nextstep3-ix86-client
nextstep3-ix86-server
aix4-r1-client
aix4-r1-server
hpux10-client
hpux10-server
w32-ix86-server
w32-ix86-client
mips-irix5-server
mips-irix5-client
osf-axp-server
osf-axp-client
```

Note: The \texttt{nextstep3-ix86}, \texttt{mips-irix5}, and \texttt{osf-axp} are not currently available in TME 10 AEF 3.0. Contact your support provider for information on the availability of these platforms.
In large distributed networks, machines are frequently grouped according to the type of work for which they are used. For example, machines in an engineering group might be used to produce CAD drawings, while those in an accounting group might be used to produce tax documents. With TME 10, you can place common configuration information for machines used for similar purposes in a centralized area. This makes it easier to access, manage, and duplicate resources.

To provide easy control over data flow to groups of systems, TME 10 provides profile-based applications. With profiles, you can develop prototypes, known as profiles, distribute them across a network, and apply them to a heterogenous set of machines. In this model, information is stored in a central location, but can also be distributed to numerous locations. You can affect many machines and system configurations with a minimal amount of time and effort.

The TME 10 AEF provides commands that allow you to customize profile-based applications, such as TME 10 User Administration. To determine if the application you want to modify is a profile-based application, refer to the customization appendix in the application’s manual.

This chapter assumes you are familiar with the basic concepts and operations of a profile-based application. See the *TME 10 Framework User’s Guide* for a complete introduction to profiles.
Profile Properties

A profile is a collection of application-specific information. The information in a profile is specific to the particular profile type. For example, a user profile might contain information such as the user name, login name, user ID, user group, and home directory for each user that is defined in the profile. A profile is actually a set of records stored in a platform-independent database format. Each profile record represents a single element of the profile. For example, in a user profile, there is a profile record for each user.

The various pieces of information stored in a profile record are called properties. In a user profile, there might be properties for the user’s name, password, user ID, and so on. TME 10 AEF allows you to create and remove custom properties on a profile’s records.

The TME 10 AEF commands that add and remove properties allow you to customize individual profiles. You cannot customize properties on a resource-wide basis.

Adding a Profile Property

The `waddprop` command adds a property to a profile. The command’s syntax is shown below.

```
waddprop profile property value
```

The `profile` parameter is the object path of the profile you want to customize. The `property` parameter is the name of the new property, and `value` is the initial value assigned to the property.

An example of the `waddprop` command is shown below.

```
% waddprop @UserProfile:engineers extension "ext. 0"
```

The example adds an `extension` property to all records in the `engineers` profile. The initial value of the property is the string `ext. 0`.

When you add a property to a profile, the application automatically adds a column to the profile’s table gadget on the profile properties window. However, the new column may not be initially visible when you open the profile properties window. If the column is not visible when you open the profile properties window, you can use the `Display`
Attributes dialog to make the column visible. To open the Display Attributes dialog, select the View->Sort->Attributes menu from the pull-down menu on the properties window.

From the Display Attributes dialog, you can select which table columns appear in the properties table, and you can control the order in which the columns appear.
Removing a Profile Property

The \textit{wrmprop} command removes a property from a profile. The command only removes properties that were added with the \textit{waddprop} command. The syntax of \textit{wrmprop} is shown below.

\texttt{wrmprop profile property}

The \textit{profile} parameter is the object path of the profile you want to customize. The \textit{property} parameter is the name of the new property.

\% \texttt{wrmprop @UserProfile:engineers extension}

The example above removes the \textit{extension} property from the \textit{engineers} profile.

Actions

Profile-based applications support a mechanism for invoking one or more \textit{actions} when a profile is distributed to its subscribers. You can use TME 10 AEF to install custom actions on a profile. An action is a shell script that is associated with creation, deletion, or modification of profile records. You can create actions that run before or after the profile is distributed. You can also can specify whether the action executes on the host where the source profile resides or on each endpoint.

Each time a record is created, removed, or changed, TME 10 adds any associated actions to its list of pending actions. Suppose you add a create action to a user profile. The action adds the new user’s name to a database configuration file. Each time a user is added to the profile, TME 10 adds the action to a list of actions to run when the profile is distributed. So, if 47 users have been added to the profile since the last distribution, the script runs 47 times — once for each record added.

Since actions run once for each record, there can be considerable performance ramifications if you plan to define actions that execute on the profile endpoints. In the example above, if the action is defined to run on the host where the profile resides, the action runs 47 times. If the action is defined to run on each endpoint, however, and you distribute the profile to 47 hosts, the action executes 2,209 times.
In format, action scripts are like normal shell scripts. Tivoli recommends that you use the Bourne shell (/bin/sh) to create action scripts. Action scripts accept parameters and must return an exit code.

Action scripts must return a zero exit code on success or a non-zero code on failure. How TME 10 handles an error depends on whether the action is run before distribution or after. There are three phases to a distribution:

1. Run the pre-distribution scripts.
2. Distribute the profile.
3. Run the post-distribution scripts.

If any of the pre-distribution scripts fail, TME 10 aborts the distribution. Any pre-distribution scripts that were not executed, including the one that failed, remain on the list of pending actions. If the distribution phase completes successfully, TME 10 runs the post-distribution action scripts. If any of these scripts fail, TME 10 does not abort the distribution. Instead, it places the failed script back on the list of pending actions and continues. If an action that is defined to run on an endpoint fails, distribution to that endpoint is aborted.

Action scripts are stored in the profile database. When a user distributes a profile, the subscribers receive copies of the scripts. Like other components of a local profile, you can override a subscriber’s copy of the action scripts with local modifications. See the TME 10 Framework User’s Guide for more information about making subscriber-level modifications to a profile.

Adding an Action

The waddaction command adds an action to a profile. The syntax of waddaction is shown below.

```
waddaction {-b | -a | -B | -A} {-c | -d | -m} profile action [args=arg1,arg2,...] < action_body
```

The `profile` parameter is the name of the profile and the `action` parameter is the name of the new action.
Actions

Note: Actions associated with modify operations can be defined on a per-attribute basis. Therefore, TME 10 AEF requires that modify actions have the same name as the profile attribute to which they are related. This restriction does not apply to create or delete actions.

The –b and –a options specify that the action will run on the source profile before or after distribution, respectively. The –B and –A options specify that the action will execute on each endpoint before or after distribution. You must include one of these four options on the command line.

The –c, –d, and –m options indicate whether the action is triggered by creation, deletion, or modification of profile records. You must include one of these options.

The args option allows you to specify a set of arguments that are passed to the action script as command line arguments. The arguments are given as a comma-separated list. To include record properties as arguments of a script, you can specify the property in $property format. If your argument list contains record properties, you must enclose the argument list in quotes.

The waddaction command reads the body of the action script from standard input. An example of waddaction is shown below.

```
% waddaction –a –d engineers remove_entry
'args=$login' < remove_entry.sh
```

The example adds a post-distribution action called remove_entry, which is associated with record deletion. The action script takes one parameter, which is the value of the record’s login property. The command reads the body of the remove_entry action from the file remove_entry.sh.

Getting an Action Definition

The wgetaction command returns the body of a given action. The command’s syntax is shown below.

```
wgetaction {–b | –a | –B | –A} {–m | –c | –d} profile action
> action_body
```

Version 3.2
The profile parameter is the name of the profile and the action parameter is the name of the action to retrieve.

The –b and –a options specify that the action will run on the source profile before or after distribution, respectively. The –B and –A options specify that the action will execute on each endpoint before or after distribution. You must include on of these four options on the command line.

The –c, –d, and –m options indicate whether the action is triggered by creation, deletion, or modification of profile records. You must include one of these options.

The command prints the body of the action script to standard output. You can redirect the output to a file.

% wgetaction –a –d engineers remove_entry > remove_entry.sh

The example above gets the body of the remove_entry script stored on the engineers profile. The body of the script is redirected to the file remove_entry.sh.

### Getting a List of Actions

The wlsactions command provides a list of actions associated with a profile. The command’s syntax is shown below.

```
wlsactions [–b | –a | –A | –B] [–c | –d | –m] profile
```

The profile parameter is the name of the profile and the action parameter is the name of the new action.

The –a, –b, –A, and –B specify the type of actions that the command should list.

<table>
<thead>
<tr>
<th>Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>–a</td>
<td>Lists all post-distribution actions that run on the source profile</td>
</tr>
<tr>
<td>–b</td>
<td>Lists all pre-distribution actions that run on the source profile</td>
</tr>
</tbody>
</table>
Actions

If you do not specify any of the above options, `wlsactions` looks for all four kinds of action.

The `-c`, `-d`, and `-m` options indicate whether the action is triggered by creation, deletion, or modification of profile records. If you do not use any of the options, `wlsactions` looks for actions of all types.

The command prints the profile’s actions to standard output, one action per line. Some examples of `wlsactions` are shown below.

```bash
% wlsactions engineers
add_entry
assign_office
change_entry
remove_entry

% wlsactions -c engineers
add_entry
assign_office
```

In the first example, `wlsactions` prints all the actions associated with the engineers profile. The second example prints the actions associated with record creation.

### Getting a List of Pending Actions

The `wgetpending` command lists the actions that are scheduled to run the next time a profile is distributed. The command’s syntax is shown below.

```
wgetpending profile
```

The `profile` parameter is the name of the profile.

<table>
<thead>
<tr>
<th>Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>Lists all post-distribution actions that run on each profile endpoint</td>
</tr>
<tr>
<td>-B</td>
<td>Lists all pre-distribution actions that run on each profile endpoint</td>
</tr>
</tbody>
</table>
The command prints the profile’s pending actions to standard output, one action per line. An example is shown below.

```
% wgetpending engineers
add_entry
assign_office
```

In the example, `add_entry` and `assign_office` actions are pending on the `engineers` profile.

### Removing an Action

The `wrmaction` command removes a custom action installed with `waddaction`. The command’s syntax is shown below.

```
wrmanction (-b | -a | -B | -A) (-c | -d | -m) profile action
```

The `profile` parameter is the name of the profile and the `action` parameter is the name of the new action.

The `–a` and `–b` options specify that the action is a pre- or post-distribution action that runs on the source profile. The `–A` and `–B` options specify that the action is a pre- or post-distribution action (respectively) that executes on each endpoint that receives the profile. You must include one of these four options on the command line.

The `–c`, `–d`, and `–m` options indicate whether the action is triggered by creation, deletion, or modification of profile records. You must include one of these options.

An example of `wrmanction` is shown below.

```
% wrmanction -a -c engineers assign_office
```

The example removes the `assign_office` action from the `engineers` profile. The action runs after distribution (`–a` option) and is associated with record creation (`–c` option).

### Profile Dialogs

Profile dialogs are the same as other kinds of dialogs. They are created with DSL, and you can use TME 10 AEF to customize them.
When you add a property to a profile, you do not need to add a column to the profile’s **Table** gadget; TME 10 automatically adds the column. However, you must add a gadget to any application dialogs where the properties value can be changed. There are a couple of guidelines that you must following when adding dialog support for a profile property.

- The **Name** attribute of the gadget must be the same as the property’s name.
- The property’s gadget must be a **Choice**, **Page**, **Text**, or **Switch** gadget.

You can edit profile dialogs in the same manner as other dialogs. See Chapter 5, “Custom Dialogs” for more information about creating custom dialogs.

If you want to provide column editing for your new property, install a dialog called **Aef_Attr_Dialog_property** as a resource-wide customization of the profile. For specific information about column editing support, and profile customizations in general, see the TME 10 AEF appendix in the user’s manual of the application you want to modify.
Putting It Together

This chapter takes the concepts that you have learned in the preceding chapters and uses them to create full-fledged, working customizations.

Example: Adding an Attribute

To add a custom attribute to a resource type, you must

- Add a dialog gadget that allows the user to change the value of the attribute.
- Define the attribute on instances of the resource type.
- Create methods that get and set the attribute’s value.

Since it can be time consuming to define the attribute on all instances at once, the first two steps are usually combined using a strategy called lazy propagation. In lazy propagation, an attribute is added to an instance the first time it is needed. For example, your methods that get and set the new attribute can first check an instance for the attribute. If the attribute does not exist, the methods can use `waddattr` to add it.

In this example, we will create a location attribute for the `ManagedNode` resource type. This attribute allows the user to note the physical location of a machine. For this new attribute, we will add a gadget to the `Managed Node Properties` dialog. We will also create two new callback methods to get and set the attribute’s value.
Dialog Gadget

The location attribute is a description of a managed node’s physical location. To support this attribute, we will add a Text gadget to the Managed Node Properties dialog.

The new, custom version of the Managed Node Properties dialog is shown below.

Use the following steps to add a custom gadget to the Managed Node Properties dialog:

1. Use wgetdialog to retrieve the dialog. The dialog is stored on the ManagedNode resource under the internal name parent_dialog.

   ```
   % wgetdialog -r ManagedNode parent_dialog \ 
   > new_parent.d
   ```
2. Reverse compile the dialog descriptor to obtain the dialog’s DSL source code.

   % rdsl new_parent.d > new_parent.dsl

   The command reverse compiles the `new_parent.d` descriptor and writes the resulting DSL source code to the file `new_parent.dsl`.

3. Modify the DSL source code to include a text gadget for the location attribute. The new lines are shown in bold in the listing below.

   1 Command Dialog
   2 {
   3   Variables
   4   {
   5     CString man_node_name;
   6     CString man_node_hostid;
   7     CString man_node_memory;
   8     CString man_node_os_name;
   9     CString man_node_os_version;
  10     CString man_node_os_release;
  11     CString man_node_location =
  12       get_location@();
  13     CString interface_list;
  14     CString interpreter;
  15   }
  16
  17 Attributes
  18 {
  19   BitmapTitle = $interpreter;
  20   Iconic = NO;
  21   Default = ok;
  22   HelpMessage = Msg(ManagedNodeCatalog,
  23     "No Help Available",17);
  24   Name = managed_node_properties;
  25   Purpose = ICONIFY;
  26   DialogStyle = PRIMARY_APPLICATION_MODAL;
  27   Title = Msg(ManagedNodeCatalog,
  28     "Managed Node",16);
  29   }

Example: Adding an Attribute

30
31 Gadgets
32 {
33
// Code not shown to save space.
34
35 Group
36 {
37 Attributes
38 {
39   Border = YES;
40   Layout = GRID;
41   Name = g0;
42   Title = Msg(ManagedNodeCatalog,
43     "Properties:",19);
44   TitlePos = TOP;
45 }
46
47 Gadgets
48 {
49 Message
50 {
51   Name = node_name_label;
52   Title = Msg(ManagedNodeCatalog,
53     "SystemName:",20);
54   GridHorizontal = 2;
55   GridVertical = 1;
56   ChildColumnAlignment = LEFT;
57 }
58
59 Message
60 {
61   Name = node_name;
62   Title = "";
63   Value = $man_node_name;
64   GridHorizontal = 4;
65   GridVertical = 1;
66   ChildColumnAlignment = LEFT;
67 }
68
69 // Gadgets omitted.
70

Example: Adding an Attribute

```plaintext
72     Text
73     {
74       Name = location;
75       Title = "Physical Location:"
76       TitlePos = LEFT;
77       Value = $man_node_location;
78       GridHorizontal = 4;
79       GridVertical = 7;
80       ChildColumnAlignment = LEFT;
81     }
82 }
83 }
84 }
85 Message
86 {
87       Title = ""
88 }
89 }
90 Group
91 {
92     // IP interface gadgets not shown.
93 }
94 }
95 Message
96 {
97     Title = ""
98 }
99 }
100 CommandButton
101 {
102     Commands =
103       set_location@$g0.location,
104       ok@$owner();
105     Name = ok;
106     Title = Msg(ManagedNodeCatalog,
107       "Update & Close",14);
108 }
109 }
110 CommandButton
111 {
112     Commands =
113       cancel@$owner();
```
Example: Adding an Attribute

```
114    Name = cancel;
115    Title = Msg(GenericCollectCat,
116        "Cancel",40);
117  }
118  }
119  }
```

4. Compile a new dialog descriptor using the modified version of the dialog source code.

```
% dsl new_parent.dsl > new_parent.d
```
The command compiles the `new_parent.dsl` DSL file and writes the dialog descriptor to `new_parent.d`.

5. Install the custom dialog descriptor using the `wputdialog` command.

```
% wputdialog –r ManagedNode –T parent_dialog \\< new_parent.d
```

Callback Methods

The `location` attribute requires a methods to get and set the attribute’s value. The Managed Node Properties dialog will use the `get_location` method to provide an initial value for the location dialog gadget. The method employs lazy propagation to create the location attribute the first time the method is called.

The listing of the `get_location` method is shown below.

```
1  #!/bin/sh
2  #
3  #
4  # Define some other helpful variables
5  #
6  SELF=`wself -l`
7  DEFAULT="Unknown"
8  CLI_EXIT_OK=0
9  CLI_EXIT_USAGE=1
10  CLI_EXIT_ERROR=2
11  #
12  #
```
Example: Adding an Attribute

15 # First, call wgetattr to see if the attribute exists.
16 #
17 VALUE=`wgetattr location @ManagedNode:$SELF`
18
19 #
20 # If the attribute does not exist, use waddattr to
21 # create it.
22 #
23 if [ $? -ne $CLI_EXIT_OK ] ; then
24  VALUE=$DEFAULT
25  waddattr -v $VALUE location @ManagedNode:$SELF
26 fi
27
28 echo $VALUE
29 exit $CLI_EXIT_OK

The method first calls wgetattr to get the value of the attribute. If the exit code from wgetattr is not CLI_EXIT_OK, the attribute does not exist. In that case, the method uses waddattr to create the attribute with the default value Unknown. Finally, the method writes the value of the attribute to standard output and exits.

The ManagedNode Properties dialog uses the set_location method to set the location attribute to the value of the location gadget. Like get_location, the set_location method uses lazy propagation to create the location attribute if it does not already exist. In this example, set_location does not really need to check for the attribute’s existence, since get_location is always called before set_location. However, it is provided for the sake of completeness and it provides a redundant system in case get_location fails.

The listing of the set_location method is shown below.

1 #!/bin/sh
2 #
3 VALUE=$1
4
5 #
6 # Define some other helpful variables
7 #
8 SELF=`wself -l`
9 CLI_EXIT_OK=0
10 CLI_EXIT_USAGE=1
Example: Adding an Attribute

11 CLI_EXIT_ERROR=2
12
13 wsetattr "$VALUE" location @ManagedNode:$SELF
14
15 #
16 # If the attribute does not exist, use waddattr to
17 # create it and exit.
18 #
19 if [ $? -ne $CLI_EXIT_OK ] ; then
20  waddattr -v "$VALUE" location @ManagedNode:$SELF
21
22  # If waddattr failed, there is a problem.
23  if [ $? -ne $CLI_EXIT_OK ] ; then
24   exit $CLI_EXIT_ERROR
25  fi
26 fi
27
28 exit $CLI_EXIT_OK

The method takes a single parameter, which is the new value of the
attribute. The method calls wsetattr to set the value. If wsetattr fails,
set_location assumes the attribute does not exist and calls waddattr
to create it. If waddattr fails, there is something seriously wrong and
set_location aborts.

Use the following steps to install the methods on the ManagedNode
resource type.

1. If you have not already created directories for your custom
   methods under the platform binary directories, create them.

   % mkdir /install_dir/bin/aix3-r2/CUSTOM
   % mkdir /install_dir/bin/dgux5/CUSTOM
   ...  
   % mkdir /install_dir/bin/uw2-ix86/CUSTOM

2. Make sure that the CUSTOM directories have mode 755.

   % chmod 755 /install_dir/bin/aix3-r2/CUSTOM

Repeat this step for each platform type in your installation.
3. Copy the `get_location.sh` and `set_location.sh` scripts to each CUSTOM directory. Repeat this step for each platform type.

   ```bash
   % cp get_location.sh /install_dir/bin/aix3-r2/CUSTOM
   % cp set_location.sh /install_dir/bin/aix3-r2/CUSTOM
   ...
   % cp get_location.sh /install_dir/bin/uw2-ix86/CUSTOM
   % cp set_location.sh /install_dir/bin/uw2-ix86/CUSTOM
   ```

4. Make sure that the copies of the scripts have mode 755. Repeat this step for each platform type.

   ```bash
   % chmod 755 /
   /install_dir/bin/aix3-r2/CUSTOM/get_location.sh
   % chmod 755 /
   /install_dir/bin/aix3-r2/CUSTOM/set_location.sh
   ...
   % chmod 755 /
   /install_dir/bin/uw2-ix86/CUSTOM/get_location.sh
   % chmod 755 /
   /install_dir/bin/uw2-ix86/CUSTOM/set_location.sh
   ```

5. Repeat steps 1–4 for every file server where TME 10 is installed.

6. Use the `wputmeth` command to install the methods.

   ```bash
   % wputmeth -r ManagedNode get_location \
   /CUSTOM/get_location.sh
   % wputmeth -r ManagedNode set_location \
   /CUSTOM/set_location.sh
   ```

---

**Example: Adding Menu Options**

To add a custom menu option to a resource type, you must

- Add the options to the **ManagedNode** resource’s pop-up menu.
- Add a **Run** menu with reboot and shutdown options to the resource’s pull-down menu.
- Create custom methods for the new options.
- Create custom validation policy for the shutdown option.
Example: Adding Menu Options

Pop-Up Menu

To add options for shutting down and rebooting a node, we will add a Run sub-menu to the ManagedNode resource’s icon pop-up menu. To be complete, we will also move the Run xterm option under the new sub-menu.

Use the following procedure to modify the ManagedNode resource type’s icon pop-up menu:

1. Use the wgetdialog command to retrieve the icon menu for one of the resource states (sunos4 is shown below).

   ```
   % wgetdialog -I sunos4-client -r ManagedNode > icon.d
   ```

2. Reverse compile the dialog descriptor to obtain the dialog’s DSL source code.

   ```
   % rdsl icon.d > new_icon.dsl
   ```

   The command reverse compiles the icon.d descriptor and writes the resulting DSL source code to the file new_icon.dsl.

3. Modify the DSL source code to include the new menu item. The new lines are shown in bold in the listing below.

   ```
   1 Command Dialog
   2 {
   3
   4 //
   5 // Icon Gadget
   6 //
   7 Icon {
   8   Name = icon;
   9   Type = ManagedNode;
   10   Commands = icon_menu;
   11 }
   12
   13 //
   14 // Icon Menu
   15 //
   16 Menu {
   17   Name = icon_menu;
   18   Title = "Managed Node";
   ```
Example: Adding Menu Options

    Button {
      Name = open_tool;
      Title = "Open ..."
      Commands = display_view();
    }

    Button {
      Name = run_properties;
      Title = "Managed Node Properties";
      Commands = show_properties();
    }

    Menu {
      Attributes {
        Name = mno_run_menu;
        Title = Run;
      }

      Contents {

        Button {
          HelpMessage = "Open an Xterm";
          Name = xterm;
          Title = "Xterm ..."
          Commands = display_xterm();
        }

        Button {
          HelpMessage = "Reboot the node";
          Name = reboot;
          Title = "Reboot"
          Commands = reboot_node@();
        }

        Button {
          HelpMessage =
Example: Adding Menu Options

61       "Shut down the node";
62       Name = shutdown;
63       Title = "Shutdown";
64      Commands =
65       shutdown_node@();
66      }
67    }
68  }
69  }
70  }

4. Compile a new dialog descriptor using the modified version of the DSL source code.

% dsl new_icon.dsl > new_parent.d

The command compiles the new_icon.dsl DSL file and writes the dialog descriptor to new_icon.d.

5. Install the custom dialog descriptor using the wputdialog command.

% wputdialog -I sunos4-client -r ManagedNode < new_icon.d

6. Repeat step 5 for every icon state that supports the managed node’s pop-up menu.

Pull-Down Menu

In addition to modifying the ManagedNode resource’s icon menu, we will also add a Run pull-down menu to the resource’s view dialog. The Run pull-down menu has the same options as the Run sub-menu of the icon pop-up menu. The new menu is shown below.

![Pull-Down Menu](image)

Use the following steps to create the custom pull-down menu:
1. Use `wgetdialog` to retrieve the view dialog.

   ```bash
   % wgetdialog -r ManagedNode view_dialog > new_view.d
   ```

2. Reverse compile the dialog descriptor to obtain the dialog’s DSL source code.

   ```bash
   % rdsl new_view.d > new_view.dsl
   ```

   The command reverse compiles the `new_view.d` descriptor and writes the resulting DSL source code to the file `new_view.dsl`.

3. Modify the DSL source code to include the new menu item. The new lines are shown in bold in the listing below.

   ```plaintext
   1 Command Dialog
   2 {
   3   Variables
   4   {
   5     CString dialog_name;
   6     CString interpreter;
   7   }
   8
   9   Attributes
   10  {
   11     BitmapTitle = $interpreter;
   12     Iconic = YES;
   13     Name = view_dialog;
   14     Title = $dialog_name;
   15  }
   16
   17 Gadgets
   18  {
   19    Collection
   20    {
   21      Name = managed_node_collection;
   22    }
   23
   24 Menu
   25  {
   26    Attributes
   27    {
   ```
Example: Adding Menu Options

28     Name = mno_system_menu;
29     Title = System;
30  }
31  
32  Contents
33  {
34     Button
35     {
36      Commands =
37        dtc_return($owner,
38        view_dialog,
39        $self);
40      HelpMessage = S_RETURN;
41      Name = return;
42      Title = Return;
43     }
44  }
45  
46  Button
47     {
48      Commands =
49        sync_launch%$owner();
50      HelpMessage = S_CLOSE;
51      Name = sync;
52      Title = "Synchronize ...";
53     }
54  
55  Button
56     {
57      Commands =
58        dtc_free($owner,
59        view_dialog,
60        $self);
61      HelpMessage = S_CLOSE;
62      Name = close;
63      Title = Close;
64     }
65  }
66  
67  Menu
68  {
69     Attributes
Example: Adding Menu Options

```plaintext
{ Name = mno_edit_menu;
  Title = Edit;
}

// The gadgets for this menu are
// not shown to conserve paper.
```

```plaintext
Menu
{
  Attributes
  {
    Name = mno_view_menu;
    Title = View;
  }

  // The gadgets for this menu are
  // not shown to conserve paper.
}
```

```plaintext
Menu
{
  Attributes
  {
    Name = mno_run_menu;
    Title = Run;
  }

  Contents
  {
    Button {
      HelpMessage = "Open an Xterm";
      Name = xterm;
      Title = "Xterm ...";
      Commands = display_xterm();
    }
    Button {
      HelpMessage = "Reboot the node";
```
Example: Adding Menu Options

```plaintext
112       Name = reboot;
113       Title = "Reboot";
114      Commands =
115       reboot_node@();
116      }
117    }
118     Button {
119       HelpMessage =
120       "Shut down the node";
121       Name = shutdown;
122       Title = "Shutdown";
123      Commands =
124       shutdown_node@();
125      }
126   }
127 }
128 }
129)
4. Compile a new dialog descriptor using the modified version of the dialog source code.

% dsl new_view.dsl > new_view.d

The command compiles the `new_view.dsl` DSL file and writes the dialog descriptor to `new_view.d`.

5. Install the custom dialog descriptor using the `wputdialog` command.

% wputdialog –r ManagedNode –T view_dialog < new_view.d

The command installs the custom version of the `ManagedNode` resource’s `view_dialog` dialog.

### Creating Callback Methods

The new menu options require the addition of two custom callback methods. The `reboot_node` method reboots the node, and the `shutdown_node` performs a shutdown.
Reboot Method

The `reboot_node` method is the callback method for the new Reboot option. The method first confirms the user’s desire to reboot the node and then calls the appropriate command based on the node’s machine type.

```bash
#!/bin/sh

# Name: reboot_node

# Description: Callback method for the reboot menu option. The method takes a single parameter, which is the machine type.

# Source the Tivoli environment
. /etc/Tivoli/setup_env.sh

# Define constants used by the method.
CLI_EXIT_OK=0
CLI_EXIT_ERROR=1
CLI_EXIT_USAGE=2
SELF='wself -l'
INTERP='winterp $SELF'

# Display a confirmation dialog.
CONFIRM = 'wdispconf -y "Yes" -n "No" \
          "Are you absolutely sure you want to reboot?"

if [ $CONFIRM = "NO" ] ; then
    exit $CLI_EXIT_OK
fi

# Invoke the reboot command on the host. Use the message passed to the method as the warning message.

for interp in $INTERP
    do
        case $interp in
            "aix3-r2")
            /etc/reboot
        esac
    done
```
Example: Adding Menu Options

```bash
37 ;;
38 "dgux5" *)
39 /sbin/reboot
40 ;;
41 "hpux9" *)
42 /etc/shutdown -r -t now $MESSAGE
43 ;;
44 "sysv4-m88k" *)
45 /etc/ucb/reboot
46 ;;
47 "solaris2" *)
48 /usr/sbin/reboot
49 ;;
50 "sunos4" *)
51 /etc/reboot
52 ;;
53 *)
54 exit CLI_EXIT_ERROR
55 ;;
56 esac
57
58 #
59 # Exit with a code indicating whether the
60 # script succeeded or failed
61 #
62 if [ $? -ne 0 ]; then
63 exit $CLI_EXIT_ERROR
64 fi
65 exit $CLI_EXIT_OK
```

### Shutdown Method

The `shutdown_node` method is the callback method for the `Shutdown` menu option. The method first validates the shutdown request. See “Policy Support” on page 11-22 for a description of the `mn_val_shutdown` policy method. The request meets policy, `shutdown_node` confirms the user’s desire to perform a shutdown. If the user confirms the request, the method invokes the appropriate command to shut down the node.
The listing of the `shutdown_node` method is shown below.

```bash
#!/bin/sh
#
# Name: shutdown_node
#
# Description: Callback method for the shutdown menu option. The method takes a single parameter, which is the type of machine.
#
# Source the Tivoli environment
. /etc/Tivoli/setup_env.sh

# Define constants used by the method.
CLI_EXIT_OK=0
CLI_EXIT_ERROR=1
CLI_EXIT_USAGE=2
SELF='wself -l'
INTERP='winterp $SELF'

# Get the result of the validation method
RESULT='wrunpolm mn_val_shutdown'

# If validation failed, exit with an error
if [ $RESULT != "TRUE" ]
  wdisperr "Policy Failure: Cannot shut down."
  exit CLI_EXIT_ERROR
fi

# Display a confirmation dialog.
CONFIRM = 'wdispconf -y "Yes" -n "No" \
  "Are you absolutely sure you want to shutdown?"
if [ $CONFIRM = "NO" ] ; then
  exit $CLI_EXIT_OK
fi
```
Example: Adding Menu Options

```bash
41 #
42 # Invoke the shutdown command on the host. Use the
43 # message passed to the method as the warning message,
44 # if it is supported by the platform.
45 #
46 #
47 case $INTERP in
48   "aix3-r2**)
49     /etc/shutdown
50     ;;
51   "dgux5**)
52     /etc/shutdown -y
53     ;;
54   "hpux9**)
55     /etc/shutdown -h -t now
56     ;;
57   "sysv4-m88k**)
58     /etc/ucb/shutdown -y -i0
59     ;;
60   "solaris2**)
61     /usr/sbin/shutdown -y
62     ;;
63   "sunos4**)
64     /etc/shutdown
65     ;;
66   *)
67     exit CLI_EXIT_ERROR
68     ;;
69 esac
70
71 #
72 # Exit with a code indicating whether the
73 # script succeeded or failed
74 #
75 if [ $? -ne 0 ]; then
76   exit $CLI_EXIT_ERROR
77 fi
78 exit $CLI_EXIT_OK
```
Installing the Custom Methods

Use the following steps to install the `shutdown_node` and `reboot_node` methods on the `ManagedNode` resource type.

1. If you have not already created directories for your custom methods, create them. These example methods encapsulate their platform-specific commands within a single version, so you can install one copy of the methods under the `default` platform type.

   ```
   % mkdir /install_dir/bin/aix3-r2/CUSTOM
   ...
   % mkdir /install_dir/bin/uw2-ix86/CUSTOM
   ```

2. Make sure that the `CUSTOM` directory has mode 755. Repeat this step for each supported platform type in your installation.

   ```
   % chmod 755 /install_dir/bin/aix3-r2/CUSTOM
   ...
   % chmod 755 /install_dir/bin/uw2-ix86/CUSTOM
   ```

3. Copy the `shutdown_node.sh` and `reboot_node.sh` scripts to the `CUSTOM` directory. Repeat this step for each platform type.

   ```
   % cp shutdown_node.sh /install_dir/bin/aix3-r2/CUSTOM
   % cp reboot_node.sh /install_dir/bin/aix3-r2/CUSTOM
   ...
   % cp shutdown_node.sh /install_dir/bin/uw2-ix86/CUSTOM
   % cp reboot_node.sh /install_dir/bin/uw2-ix86/CUSTOM
   ```

4. Make sure that the copies of the scripts have mode 755. Repeat this step for each platform type in your installation.

   ```
   % chmod 755 \
   /install_dir/bin/aix3-r2/CUSTOM/shutdown_node.sh
   % chmod 755 \n   /install_dir/bin/aix3-r2/CUSTOM/reboot_node.sh
   ...
   % chmod 755 \
   /install_dir/bin/uw2-ix86/CUSTOM/shutdown_node.sh
Example: Adding Menu Options

% chmod 755\
/install_dir/bin/uw2-ix86/CUSTOM/reboot_node.sh

5. Repeat steps 1–4 for every file server where TME 10 is installed.

6. Use the `wputmeth` command to install the methods.

% wputmeth -r ManagedNode shutdown_node \
-u root -g sys /CUSTOM/shutdown_node.sh

% wputmeth -r ManagedNode reboot_node \
-u root -g sys /CUSTOM/reboot_node.sh

Since the methods run system commands that require the root user, you must install the methods with root privilege. The `-u` and `-g` options set the effective user and group privileges of the methods.

Policy Support

The `shutdown_node` method has a validation policy associated with it. For this example, we will implement a policy that prohibits shutting down a machine between 8 a.m. and 5 p.m.

Use the following steps to create the custom policy method:

1. Create the policy method script. The listing is shown below.

```bash
#!/bin/sh

# Get the hour field from the system date
HOUR='\`/bin/date +%H\`

# If it is between 8 a.m. and 5 p.m.,
# do not allow the reboot
if [ $HOUR -ge 8 -a $HOUR -le 17 ] ; then
    echo "FALSE"
else
    echo "TRUE"
fi
```

The method uses the system `date` command to get the current hour. If the hour is between eight and seventeen, `mn_val_shutdown` returns `FALSE`. Otherwise, the method returns `TRUE`.
Example: Adding Menu Options

2. Install the policy method on the ManagedNode resource type.

    ```
    % waddpolm -i default -v ManagedNode mn_val_shutdown \
    /work/scripts/shutdown.sh
    ```
Example: Adding Menu Options
Dialog Descriptors

This appendix contains lists of the dialog descriptors and icon states that are supported by each resource type in the TME 10 Framework. To get information about dialog descriptors and icon states of application resources, refer to the AEF appendix of the application’s manual.

Dialog Descriptors

This section lists the dialog descriptors defined for each resource type in the TME 10 Framework. To get information about dialogs associated with application resources, refer to the AEF appendix of the application’s manual.

In the tables in this section, the dialogs’ actual titles are shown whenever possible. These are shown in boldface type. Where it is not possible to show the dialog title (because it does not have one, or because it is context-sensitive), a description of the dialog is given in place of the title. These are listed in regular font.

Administrator

The table below shows the dialogs supported by the Administrator resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Policy Region</td>
<td>create_policy_region</td>
</tr>
</tbody>
</table>
## Dialog Descriptors

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TME 10 desktop</td>
<td>desktop</td>
</tr>
<tr>
<td>Edit Notice Groups</td>
<td>edit_notice_groups</td>
</tr>
<tr>
<td>Exit Maintenance Mode</td>
<td>end_maint</td>
</tr>
<tr>
<td>Administrator icon menu</td>
<td>icon</td>
</tr>
<tr>
<td>Set Login Names</td>
<td>logins</td>
</tr>
<tr>
<td>TMR Maintenance Mode</td>
<td>maint</td>
</tr>
<tr>
<td>Administrator Properties</td>
<td>properties</td>
</tr>
<tr>
<td>Set Resource Roles</td>
<td>resource_roles</td>
</tr>
<tr>
<td>Set TMR Roles</td>
<td>tmr_roles</td>
</tr>
</tbody>
</table>

## AdministratorCollection

The table below shows the dialogs supported by the AdministratorCollection resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Administrator</td>
<td>create_administrator</td>
</tr>
<tr>
<td>Administrator collection view</td>
<td>desktop</td>
</tr>
<tr>
<td>Edit Notice Groups</td>
<td>edit_notice_groups</td>
</tr>
<tr>
<td>Administrator collection icon menu</td>
<td>icon</td>
</tr>
<tr>
<td>Set Login Names</td>
<td>logins</td>
</tr>
<tr>
<td>Set Notice Groups</td>
<td>notice_groups</td>
</tr>
<tr>
<td>Set Resource Roles</td>
<td>resource_roles</td>
</tr>
<tr>
<td>Set TMR Roles</td>
<td>tmr_roles</td>
</tr>
</tbody>
</table>
### BulletinBoard

The table below shows the dialogs supported by the **BulletinBoard** resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confirm</td>
<td>confirm</td>
</tr>
<tr>
<td>Bulletin board icon menu</td>
<td>icon</td>
</tr>
<tr>
<td>Filter Notices</td>
<td>message_filter</td>
</tr>
<tr>
<td>Notice Group Messages</td>
<td>message_list</td>
</tr>
<tr>
<td>Sort Notices</td>
<td>message_sort</td>
</tr>
<tr>
<td>Notice Message Viewer</td>
<td>message_viewer</td>
</tr>
<tr>
<td>Read Notices...</td>
<td>select_group</td>
</tr>
</tbody>
</table>

### Job

The table below shows the dialogs supported by the **Job** resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Job</td>
<td>edit_job</td>
</tr>
<tr>
<td>Host &amp; path dialog</td>
<td>host_and_path</td>
</tr>
<tr>
<td>Job results dialog</td>
<td>job_output</td>
</tr>
</tbody>
</table>

### ManagedNode

The table below shows the dialogs supported by the **ManagedNode** resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add IP Interface</td>
<td>add_dialog</td>
</tr>
<tr>
<td>Edit IP Interface</td>
<td>edit_dialog</td>
</tr>
</tbody>
</table>
The table below shows the dialogs supported by the **PCManagedNode** resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managed node view dialog</td>
<td>execute_task</td>
</tr>
<tr>
<td>Host &amp; path dialog</td>
<td>host_and_path</td>
</tr>
<tr>
<td>Managed node icon menu</td>
<td>icon</td>
</tr>
<tr>
<td>Managed Node</td>
<td>parent_dialog</td>
</tr>
<tr>
<td>Task results dialog</td>
<td>task_output</td>
</tr>
<tr>
<td>Client/server toggle</td>
<td>toggle_state_dialog</td>
</tr>
<tr>
<td>Execute task</td>
<td>view_dialog</td>
</tr>
<tr>
<td>Display/Change IP Address</td>
<td>change_ip_address_dialog</td>
</tr>
<tr>
<td>DOS Host</td>
<td>dos_props_dialog</td>
</tr>
<tr>
<td>NetWare Host</td>
<td>netware_props_dialog</td>
</tr>
<tr>
<td>Client/Server icon Toggle</td>
<td>toggle_state_dialog</td>
</tr>
<tr>
<td>Windows Host</td>
<td>windows_props_dialog</td>
</tr>
<tr>
<td>Add Hosts... (create)</td>
<td>pc_add_host</td>
</tr>
<tr>
<td>Create PC Managed Nodes (create)</td>
<td>pc_create</td>
</tr>
<tr>
<td>Read Host List From File (create)</td>
<td>pc_read_host_list</td>
</tr>
</tbody>
</table>
PolicyRegion

The table below shows the dialogs supported by the PolicyRegion resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create policy region</td>
<td>create_policy_region</td>
</tr>
<tr>
<td>Policy region view</td>
<td>desktop</td>
</tr>
<tr>
<td>Set managed resources</td>
<td>edit_classes</td>
</tr>
<tr>
<td>Managed mesource policies</td>
<td>edit_policies</td>
</tr>
<tr>
<td>Policy region properties</td>
<td>edit_properties</td>
</tr>
<tr>
<td>Policy region icon menu</td>
<td>icon</td>
</tr>
</tbody>
</table>

ProfileManager

The table below shows the dialogs supported by the PolicyRegion resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Profile Manager</td>
<td>create</td>
</tr>
<tr>
<td>Create Profiles</td>
<td>create_profile</td>
</tr>
<tr>
<td>Delete Profiles</td>
<td>delete</td>
</tr>
<tr>
<td>Edit Profile Manager</td>
<td>edit</td>
</tr>
<tr>
<td>Host &amp; path dialog</td>
<td>host_and_path</td>
</tr>
<tr>
<td>Profile manager view</td>
<td>main</td>
</tr>
<tr>
<td>Distribute Profiles</td>
<td>push</td>
</tr>
<tr>
<td>Profile manager icon menu</td>
<td>profile_manager_icon</td>
</tr>
<tr>
<td>Get New Copy of Profiles</td>
<td>pull</td>
</tr>
<tr>
<td>Show Subscribers</td>
<td>subscribers</td>
</tr>
</tbody>
</table>
### Scheduler

The table below shows the dialogs supported by the **Scheduler** resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Subscriptions</td>
<td>subscriptions</td>
</tr>
<tr>
<td>Task results dialog</td>
<td>task_output</td>
</tr>
<tr>
<td>Unsubscribe Profiles</td>
<td>unsubscribe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td>find</td>
</tr>
<tr>
<td>Scheduler icon menu</td>
<td>icon</td>
</tr>
<tr>
<td>Notice</td>
<td>notice</td>
</tr>
<tr>
<td>Select Notice Groups</td>
<td>notice_browser</td>
</tr>
<tr>
<td>Add Scheduled Job</td>
<td>sched_add</td>
</tr>
<tr>
<td>Browse Scheduled Jobs</td>
<td>sched_browse</td>
</tr>
<tr>
<td>Edit Scheduled Job</td>
<td>sched_edit</td>
</tr>
<tr>
<td>Set Retry/Cancel Options</td>
<td>sched_expert</td>
</tr>
<tr>
<td>Scheduler Status</td>
<td>sched_popup</td>
</tr>
<tr>
<td>Browse Table Attributes</td>
<td>set_fields</td>
</tr>
<tr>
<td>Sort</td>
<td>sort_records</td>
</tr>
</tbody>
</table>
**TaskLibrary**

The table below shows the dialogs supported by the TaskLibrary resource type.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Descriptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Job...</td>
<td>create_job</td>
</tr>
<tr>
<td>Create Task...</td>
<td>create_task</td>
</tr>
<tr>
<td>Edit Job</td>
<td>edit_job</td>
</tr>
<tr>
<td>Edit Task...</td>
<td>edit_task</td>
</tr>
<tr>
<td>Execute Task...</td>
<td>execute_task</td>
</tr>
<tr>
<td>Executable host &amp; path dialog</td>
<td>host_and_path</td>
</tr>
<tr>
<td>List of jobs to execute</td>
<td>job_list</td>
</tr>
<tr>
<td>Create Task Library (create)</td>
<td>make_task_library</td>
</tr>
<tr>
<td>Output host &amp; path dialog</td>
<td>save_host_and_path</td>
</tr>
<tr>
<td>List of tasks to execute</td>
<td>task_list</td>
</tr>
<tr>
<td>Task results dialog</td>
<td>task_output</td>
</tr>
<tr>
<td>Task library view</td>
<td>view_dialog</td>
</tr>
</tbody>
</table>

**Icon States**

This section contains the icon states associated with each resource type in the TME 10 Framework. To get information about icon states of application resources, refer to the AEF appendix of the application’s manual.
**Administrator**

The table below lists the icon states defined for the Administrator resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Administrator icon</td>
</tr>
</tbody>
</table>

**AdministratorCollection**

The table below lists the icon states defined for the AdministratorCollection resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Administrator collection icon</td>
</tr>
</tbody>
</table>

**BulletinBoard**

The table below lists the icon states defined for the BulletinBoard resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Bulletin board icon with notices</td>
</tr>
<tr>
<td>generic_icon_polling</td>
<td>Empty bulletin board icon</td>
</tr>
</tbody>
</table>

**Job**

The table below lists the icon states defined for the Job resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Job icon</td>
</tr>
</tbody>
</table>
### ManagedNode

The table below lists the icon states defined for the ManagedNode resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix3-r2-client</td>
<td>AIX 3.2.5 client icon</td>
</tr>
<tr>
<td>aix3-r2-server</td>
<td>AIX 3.2.5 server icon</td>
</tr>
<tr>
<td>aix4-r1-client</td>
<td>AIX 4.1 client icon</td>
</tr>
<tr>
<td>aix4-r1-server</td>
<td>AIX 4.1 server icon</td>
</tr>
<tr>
<td>dgux5-client</td>
<td>DG/UX client icon</td>
</tr>
<tr>
<td>dgux5-server</td>
<td>DG/UX server icon</td>
</tr>
<tr>
<td>hpux9-client</td>
<td>HP-UX 9.x client icon</td>
</tr>
<tr>
<td>hpux9-server</td>
<td>HP-UX 9.x server icon</td>
</tr>
<tr>
<td>hpux10-client</td>
<td>HP-UX 10 client icon</td>
</tr>
<tr>
<td>hpux910-server</td>
<td>HP-UX10 server icon</td>
</tr>
<tr>
<td>sunos4-client</td>
<td>SunOS client icon</td>
</tr>
<tr>
<td>sunos4-server</td>
<td>SunOS server icon</td>
</tr>
<tr>
<td>solaris2-client</td>
<td>Solaris client icon</td>
</tr>
<tr>
<td>solaris2-server</td>
<td>Solaris server icon</td>
</tr>
<tr>
<td>sysv4-att-client</td>
<td>AT&amp;T 3000 client icon</td>
</tr>
<tr>
<td>sysv4-att-server</td>
<td>AT&amp;T 3000 server icon</td>
</tr>
<tr>
<td>sysv4-m88k-client</td>
<td>Motorola 88k client icon</td>
</tr>
<tr>
<td>sysv4-m88k-server</td>
<td>Motorola 88k server icon</td>
</tr>
</tbody>
</table>
Icon States

PCManagedNode

The table below lists the icon states defined for the PCManagedNode resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dos_client</td>
<td>DOS client icon</td>
</tr>
<tr>
<td>dos_server</td>
<td>DOS server icon</td>
</tr>
<tr>
<td>netware_client</td>
<td>Novell NetWare client icon</td>
</tr>
<tr>
<td>netware_server</td>
<td>Novell NetWare server icon</td>
</tr>
<tr>
<td>windows_client</td>
<td>Microsoft Windows client icon</td>
</tr>
<tr>
<td>windows_server</td>
<td>Microsoft Windows server icon</td>
</tr>
</tbody>
</table>

PolicyRegion

The table below lists the icon states defined for the PolicyRegion resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Policy region icon</td>
</tr>
</tbody>
</table>

ProfileManager

The table below lists the icon states defined for the ProfileManager resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Profile manager icon</td>
</tr>
</tbody>
</table>
Scheduler

The table below lists the icon states defined for the **ProfileManager** resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Scheduler icon</td>
</tr>
</tbody>
</table>

TaskLibrary

The table below lists the icon states defined for the **TaskLibrary** resource type.

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>library</td>
<td>Task library icon</td>
</tr>
<tr>
<td>task</td>
<td>Task icon</td>
</tr>
</tbody>
</table>
NAME
dsl

PURPOSE
Command to create a dialog descriptor from a DSL source file.

SYNOPSIS
[-u] [-V] [sourcefile]

DESCRIPTION
The dsl invokes the DSL compiler. The DSL compiler takes a DSL source file and compiles it into a binary dialog descriptor.

Arguments
  -D name       Defines symbol name to have the value def. If def is omitted, name is defined to have a null value. The -D option is valid only if -P or -E is specified. It may be given multiple times.
  -e            Suppresses informative messages, warning messages, and error messages.
  -E path       Instructs the DSL compiler to run its input through the preprocessor specified by path but not to compile the result. The preprocessor output is written to the output file specified by the -o option, or to standard output if the -o option is omitted. The DSL compiler automatically defines the symbol dsl for the preprocessor when the -E option is given. If path is omitted, the default is /lib/cpp. The -E option cannot be used with -P.
  -P path       Causes the DSL compiler to run the input file through the preprocessor specified by path, prior to compilation. If omitted, the default is /lib/cpp. The DSL compiler automatically defines the symbol dsl for the preprocessor when -P is specified.
-help  Prints a short usage message. If this option is present, all other options are ignored.

-#dir  Specifies the path of a directory to be searched for files specified by the #include directive in the DSL file. This parameter is passed to the specified preprocessor and may be given multiple times. It is valid only if the -E or -P option is specified.

-o outfile  Specifies the name of the output file. If omitted, the DSL compiler writes to standard output.

-V  Prints the version number of the DSL compiler. If this option is present, all other options are ignored.

sourcefile  Specifies the name of the file to be compiled. If sourcefile does not exist, the DSL compiler attempts to compile sourcefile.dsl instead. If the sourcefile parameter is omitted, the DSL compiler reads from standard input.

SEE ALSO

rdsl
NAME

rdsl

PURPOSE

Reverse compiles a DSL dialog descriptor.

SYNOPSIS

rdsl descriptor

DESCRIPTION

The rdsl command reverse compiles a binary dialog descriptor and produces the DSL source code. The DSL source code is ASCII text.

The descriptor is the file name that contains the binary dialog descriptor. The command writes the resulting DSL source code to standard output.

Although rdsl converts a compiled dialog to DSL source code, it is not identical to the original. The following differences can occur between the reverse compiled version and the original:

- The reverse compiled version does not retain the comments included in the original.
- The spacing can be different.
- The ordering of the attributes can be different.

SEE ALSO

dsl
NAME

waddaction

PURPOSE

Adds a custom action to a profile.

SYNOPSIS

waddaction {–a | –b | –A | –B} {–c | –d | –m} profile action
[args=arg1, arg2, ...] < action_body

DESCRIPTION

Profile-based applications support a mechanism for invoking one or more actions when a profile is distributed to its subscribers. You can use TME 10 AEF to install custom actions on a profile. An action is a shell script that is associated with creation, deletion, or modification of profile records. You can also specify whether or not the action runs before or after the actual distribution.

The waddaction command adds a new action to a profile. The command reads the body of the action script from its standard input.

You must specify whether the action script is to run before or after distribution and whether the script executes on the host where the profile resides or on each endpoint. You must use either the –a, –b, –A, or –B option.

Actions are associated with an event (creation, deletion, or modification of profile records). Use one of the following options to associate your new action with an event: –c, –d, or –m.

Actions scripts can accept arguments. The args option allows you to specify a set of arguments that are passed to the action script when it executes. The arguments are given as a comma-separated list. To include record properties as arguments of a script, specify the property in $property format. If your argument list contains record properties, enclose the argument list in quotes to prevent the shell from interpreting the property as an environment variable.

Authorization

super
Arguments

–a  Specifies that the script is to be run on the host where the profile resides after the profile is distributed to its subscribers.

–A  Specifies that the script is to be run on each endpoint after the profile is distributed to its subscribers.

–b  Specifies that the script is to be run before the profile is distributed to its subscribers.

–B  Specifies that the script is to be run on each endpoint before the profile is distributed to its subscribers.

–c  Associates the action with the creation of profile records.

–d  Associates the action with the deletion of profile records.

–m  Associates the action with the modification of profile records.

args...  A list of arguments that are passed to the action script. The argument list must be preceded by the args= keyword. Each argument in the list must be delimited by a comma.

EXAMPLES

% waddaction -a -d engineers remove_entry "$login_name" < remove_entry.sh

The example above adds a post-distribution action called remove_entry, which is associated with record deletion. The action script takes one parameter, which is the value of the record’s login_name property. The command reads the body of the remove_entry action from the file remove_entry.sh.

SEE ALSO

wgetaction, wlsactions, wrmaction
waddattr

NAME

waddattr

PURPOSE

Adds a custom attribute to one or more instances of a resource.

SYNOPSIS

waddattr [-v value] attribute label...

DESCRIPTION

The waddattr command adds a new attribute to a specified set of resource instances. You can only add attributes that are ASCII strings. To add attributes of other types, you must use the TME 10 Advanced Development Environment (TME 10 ADE).

Authorization

super

Arguments

-v value An optional initial value for the attribute.
attribute The name of the attribute to add.
label... The object paths of one or more resource instances. Each path must be in the form type@label or /distinguished/[type:]label... The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy.

EXAMPLES

The following example command adds a new attribute, location, to an instance of the ManagedNode resource type.

waddattr -v "Building 12" location @ManagedNode:moab

SEE ALSO

wgetattr, wrmattr, wsetattr
NAME

waddpolm

PURPOSE

Adds a custom policy method to a resource type.

SYNOPSIS

waddpolm [–t tmr... | –T] [–d | –v] [-c value | -C] resource method

DESCRIPTION

The waddpolm command adds a new policy method to a specified resource type. The method is added to all policies defined for the resource type within the current TMR. You can also use waddpolm to install policy methods in remote TMR.

The method parameter is the name by which the new method is referenced from within TME 10. The method can be a constant value (called an intrinsic method) or a method script. To create an intrinsic method, use the -c or -C options. If you do not use one of these options, waddpolm assumes the method is a script and reads the script from standard input.

It is important that the names of your policy methods be unique within a resource type. The names must be unique across both types of policy methods (you cannot have a defaults policy method and a validation policy method with the same name).

Authorization

super

Arguments

–t tmr... Installs the method on the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where waddpolm is run. You can use –t more than once to install the method in several TMRs.
waddpolm

-\(T\) Installs the method on the resource in the current TMR and all connected TMRs. This option cannot be used if -t is specified.

-\(d\) Indicates that the new policy method is a default policy.

-\(v\) Indicates that the new policy method is a validation policy.

-\(e\ value\) Indicates that the method is a constant value. The value parameter specifies the value.

-\(C\) Indicates that the method is a constant value. waddpolm reads the value from standard input.

resource The resource type on which to install the method.

method The name of the new policy method.

EXAMPLES

The following example command adds a new policy method, \texttt{mn\_val\_reboot}, to the \texttt{ManagedNode} class in the local TMR and all connected TMRs.

\texttt{% waddpolm \(-T \-d ManagedNode mn\_val\_reboot \< \}/work/scripts/mn\_val\_reboot.sh\fP}

SEE ALSO

wrmpolm, wrunpolm
NAME

waddprop

PURPOSE

Adds a custom property to a profile.

SYNOPSIS

waddprop profile property value

DESCRIPTION

The waddprop command adds a custom property to a profile. This is similar to adding an attribute to a resource instance.

With some types of profiles, the new property automatically appears as a column of the profile's table gadget on the desktop. On other profile types, you must manually add a new column for the new property. Consult the documentation of each profile-based application for more information.

Authorization

super

Arguments

profile The name of the profile to which you are adding the property.

property The name of the new property.

value Initial value of the new property.

EXAMPLES

The following example command adds a new property, extension, to the engineers profile. The initial value of the property is the string ext 0.

% waddprop @UserProfile:engineers extension "ext. 0"

SEE ALSO

wrmprop
NAME
  wconbitmap

PURPOSE
  Combines XBM and PPM bitmaps into a single XPM bitmap.

SYNOPSIS
  wconbitmap ppm ppm_color xbm_bw xbm_mask > xpmfile

DESCRIPTION
  The wconbitmap command allows you to convert icons into the XPM format required by TME 10.

  If your favorite bitmap editor does not support XPM, use the procedure below to convert your bitmaps into the format required by TME 10.

  1. Create a color version of the bitmap.
  2. Create a black and white version of the bitmap.
  3. Create a mask (reverse) of the bitmap. The mask is only used with icon bitmaps.
  4. Convert the bitmap to portable pixmap (PPM) format.
  5. Convert the black and white, and mask versions into X11 bitmap (XBM) format.
  6. Use the wconbitmap command to combine the PPM and XBM bitmaps into a single XPM bitmap.

  To convert bitmaps into PPM and XBM formats, TME 10 includes a public domain package of filters called PBMPLUS. PBMPLUS, created by Jef Poskanzer, can convert many common bitmap formats.

  The wconbitmap command converts the three files into a single XPM file. The command writes the XPM file to standard output; you can redirect this output to a file.

Authorization
  super
wconbitmap

Arguments

- **ppm_color**: The file name of the color version of the bitmap.
- **xbm_bw**: The file name of the black and white version of the bitmap.
- **xbm_mask**: The file name of the bitmap mask (reverse video image).

SEE ALSO

wgetbitmap, wputbitmap
NAME

wdispconf

PURPOSE

Displays a confirmation dialog.

SYNOPSIS

wdispconf [–n no-label] [–y yes-label] message

DESCRIPTION

The wdispconf command displays a two-choice dialog on the desktop where the method that invoked the command is running. The command returns information about whether the user selected the "affirmative" button or the "negative" button.

The command prints the string "YES" to standard output if the user selects the affirmative button. If the user selects the negative button, wdispconf prints "NO" to standard output.

Authorization

super

Arguments

–n no-label Specifies an optional label for the negative button. The command uses a default label if you do not use the –n option. The value of the default label is locale-dependent.

–y yes-label Specifies an optional label for the affirmative button. The command uses a default label if you do not use the –y option. The value of the default label is locale-dependent.

message The text of the message that is to appear in the dialog. The text must be pre-formatted.

EXAMPLES

% wdispconf –y "Yes" –n "No" "Are you sure you want to reboot?"
SEE ALSO

wdisperr, wdispmsg, wstatusline

NAME

wdisperr

PURPOSE

Displays a Motif-style error dialog.

SYNOPSIS

wdisperr message

DESCRIPTION

The wdisperr command displays a simple dialog that contains a description of an error or system failure. The dialog displays on the desktop on which the method that invoked wdisperr is running. The dialog has a single button, Dismiss, and displays the Motif error symbol.

Authorization

super

Arguments

message  The text of the message that is to appear in the dialog.
          The text must be pre-formatted.

EXAMPLES

% wdisperr "Could not complete the operation"

SEE ALSO

wdispconf, wdispmsg, wstatusline
NAME

wdispmsg

PURPOSE

Displays a Motif-style message dialog.

SYNOPSIS

wdispmsg message

DESCRIPTION

The **wdispmsg** command a simple dialog that contains information for the user. The dialog displays on the desktop where the method that invoked **wdispmsg** is running. The dialog has a single button, **Dismiss**.

Authorization

super

Arguments

*message* The text of the message that is to appear in the dialog. The text must be pre-formatted.

EXAMPLES

% wdispmsg "Successfully removed the selected users."

SEE ALSO

wdispconf, wdisperr, wstatusline
NAME

wformatlist

PURPOSE

Formats its arguments into a string that can be assigned to the Choices attribute of a Choice or List gadget.

SYNOPSIS

wformatlist [list-item]...

DESCRIPTION

wformatlist formats all its list-item arguments into a string that can be used with the Choice attribute of a Choice or List gadget. The command writes the formatted string to standard output. wformatlist is typically used in an AEF shell method to supply list-valued arguments to wpostdialog.

Authorization

super

Arguments

list-item An item in a Choice or List gadget. You may specify multiple list-item arguments.

EXAMPLES

The dialog below, my_dlg, contains a dialog variable that is assigned to a Choices attribute.

Command Dialog {
    Name = my_dlg;
    Variables {
        CString choices;
    }
    Choice {
        Name = c_list;
        Choices = $choices;
    }
}
To bind and post `my_dlg`, one could define the following AEF shell method, which calls `wformatlist` to supply a value for the `choices` dialog variable.

```bash
#!/bin/sh
...

wpostdialog my_dlg choices 'wformatlist choice1 choice2 choice3'
```

**SEE ALSO**

`wformatvalue`, `wpostdialog`
wformatvalue

NAME

wformatvalue

PURPOSE

Formats its arguments into a string that can be assigned to the Value attribute of a Choice or List gadget.

SYNOPSIS

wformatvalue [list-item]...

DESCRIPTION

wformatvalue formats all its list-item arguments into a string that can be used with the Value attribute of a Choice or List gadget. The Value attribute of a Choice or List gadget contains the set of elements that are currently selected.

wformatlist is typically used in an AEF shell method to supply list-valued arguments to wsetvalue.

The command prints the formatted string to standard output.

Authorization

super

Arguments

list-item An item in a Choice or List gadget. You may specify multiple list-item arguments.

EXAMPLES

The dialog below, my_dlg, contains a dialog variable that is assigned to a Choices attribute.

Command Dialog {
    Name = my_dlg;

    Variables {
        CString choices;
    }

    Choice {
        Name = c_list;
        Choices = $choices;
    }
}
The AEF shell method shown below posts `my_dlg` and populates the choice gadget using `wformatlist`. The shell method then uses `wformatvalue` with `wsetvalue` to set the initial value of the choices gadget.

```bash
#!/bin/sh
...

# Post the dialog and populate the Choice gadget
wpostdialog my_dlg choices 'wformatlist choice1 choice2 choice3'

# Set choice2 as the row initially selected in the gadget
wsetvalue my_dlg choices 'wformatvalue choice2'
```

**SEE ALSO**

`wformatlist`, `wpostdialog`
NAME
wgetaction

PURPOSE
Returns the body of an action script.

SYNOPSIS
wgetaction {–a | –b | –A | –B} {–c | –d | –m} profile action >
action_body

DESCRIPTION
The wgetaction command retrieves the body of an action script from a given profile. The action must have been installed with waddaction.
You must specify whether the action script runs before or after distribution and whether the script executes on the host where the profile resides or on each endpoint. You must use either the –a, –b, –A, or –B option.
Actions are associated with an event (creation, deletion, or modification of profile records). Use one of the following options to indicate with which event the action is associated: –c, –d, or –m.
The command prints the body of the action script to standard output. To save a copy of the script, redirect standard input to a file.

Authorization
super

Arguments
–a Specifies that the script runs on the host where the profile resides after the profile is distributed to its subscribers.
–A Specifies that the script runs on each endpoint after the profile is distributed to its subscribers.
–b Specifies that the scripts runs before the profile is distributed to its subscribers.
wgetaction

-B Specifies that the scripts runs on each endpoint before the profile is distributed to its subscribers.

-d Specifies that the action is associated with the deletion of profile records.

-m Specifies that the action is associated with the modification of profile records.

EXAMPLES

% wgetaction -a -d engineers remove_entry > remove_entry.sh

The example above gets the body of a post-distribution action called remove_entry. The action is associated with record deletion. The command redirects the body of the remove_entry action to the file remove_entry.sh.

SEE ALSO

waddaction, wlsactions, wrmaction
NAME
wgetattr

PURPOSE
Gets the value of a custom attribute.

SYNOPSIS
wgetattr attribute label

DESCRIPTION
The wgetattr command returns the value of a custom attribute for a given resource instance. You can only use wgetattr to retrieve attributes that were added with waddattr.

The command writes the attribute’s value to standard output. If the specified attribute does not exist, the command’s standard output is undefined.

Authorization
super

Arguments
attribute The name of the attribute to get.
label The object path of an instance. The path must be in the form type@label or /distinguished[type:]label... . The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy.

EXAMPLES
The following example command gets the value of the location attribute from an instance of the ManagedNode resource type.

wgetattr location @ManagedNode:moab

SEE ALSO
waddattr, wrmattr, wsetattr
NAME

wgetbitmap

PURPOSE

Retrieves a bitmap stored on a given resource or instance.

SYNOPSIS

wgetbitmap –r resource [–t tmr] bitmap-name > bitmap.xpm
wgetbitmap –l label bitmap > bitmap.xpm
wgetbitmap {–I state | –x} –r resource [–t tmr] > bitmap.xpm

DESCRIPTION

The wgetbitmap command retrieves a bitmap stored on a specific resource or instance. Dialogs in TME 10 make heavy use of bitmaps, since they provide an effective and efficient means of displaying information. The TME also uses bitmaps to represent instances as icons on the TME 10 desktop.

If you are customizing an icon bitmap on an application that uses the TME 10 transition presentation object, use the –x option. A pre-2.0 application running under TME 2.0 does not store its icon bitmap and dialog on its presentation objects. Instead, the application stores them on a presentation object provided by TME 10. This presentation object, called the transition presentation object, is shared by all “transition” applications installed in the TMR. Refer to the application documentation to determine how the application you are modifying stores its icon bitmaps and dialogs.

The TME Desktop uses bitmaps and icons that are stored in the X pixmap (XPM) format. The XPM format is used extensively in X Windows, since it allows an image to be displayed on monochrome, grey scale, or color displays.

If you want to use your favorite bitmap editor and it does not support XPM, you can still create TME bitmaps and icons. See the wconbitmap reference page for a procedure to convert bitmaps into the format required by TME 10.
wgetbitmap writes the bitmap file to standard output. You can redirect this output to a file.

Authorization

super

Arguments

–l state  Retrieves the icon bitmap for the given resource state. You cannot use this option with the –x option.

–x  Looks for the application’s icon bitmap on the TME 10 transition presentation object. You cannot use this option with the –I option.

–l label  Retrieves a custom bitmap installed on the given instance. The instance’s object path must be in the form @type:label or /distinguished[type:]label... . The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy. The first form can be used to reference instances in remote TMRs, since instances registered with the name registry in one TMR are known to other TMRs.

–r resource  Specifies the type of resource. This option is not used when getting a bitmap from an instance. When –r is used by itself, wgetbitmap looks for a custom version of the bitmap for the resource. If there is a custom version, wgetbitmap returns it. Otherwise, wgetbitmap returns the original version of the bitmap.

–t tmr  Retrieves the bitmap from the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where wgetbitmap is run.

bitmap  Name of the bitmap to retrieve.
EXAMPLES

This example retrieves the sunos4 bitmap from the ManagedNode resource.

% wgetbitmap -r ManagedNode sunos4 > sunos.xpm

SEE ALSO

wlsbitmap, wputbitmap, wrmbitmap
wgetdialog

NAME
wgetdialog

PURPOSE
Retrieves the binary dialog descriptor of a dialog.

SYNOPSIS
wgetdialog –r resource [–s] [–t tmr] dialog-name > dialog.d
wgetdialog –l label dialog-name > dialog.d
wgetdialog {–C | –c policy} –r resource [–t tmr] dialog-name >
dialog.d
wgetdialog {–I state | –x} –r resource [–t tmr]

DESCRIPTION
The wgetdialog command retrieves the binary dialog descriptor of a
dialog. You can then use the rdsl reverse compiler to get the DSL
source code for the dialog.

You can use the wgetdialog command to retrieve descriptors for
dialogs installed on an entire resource, custom dialogs installed on
specific instances, and create dialogs. The command can also retrieve
dialog descriptors that define the pop-up menus associated with a
desktop icons.

wgetdialog writes the dialog descriptor to standard output. You can
redirect this output to a file or pipe it directly to the rdsl reverse
compiler.

You can also use wgetdialog to retrieve dialog descriptors associated
with icons. To do this, use the –I option and specify the icon state
whose dialog you want to retrieve. If you are customizing an icon
bitmap on an application that uses the TME 10 transition presentation
object, use the –x option instead of –I. A pre-2.0 application running
under TME 2.0 does not store its icon bitmap and dialog on its
presentation object. Instead, the application stores them on a
presentation object provided by TME 10. This presentation object,
called the transition presentation object, is shared by all “transition”
applications installed in the TMR. Refer to the application
documentation to determine how the application you are working with
stores its icon bitmaps and dialogs.

Authorization

super

Arguments

--C
Retrieves a create dialog for a specific resource type. If the create dialog has been customized for all
defaults policies of the resource, wgetdialog returns the custom version. Otherwise, wgetdialog returns the
original version.

--c policy
Retrieves a create dialog from a specific defaults policy. If there is an individual customization for the
dialog, wgetdialog returns it. If there is not an individual customization, wgetdialog looks for a
version customized for all defaults policies. If there is no resource-wide customization, wgetdialog exits
with an error.

--I state
Retrieves the icon dialog descriptor for the given resource state.

--x
Looks for the application’s icon dialog on the TME 10 transition presentation object. You cannot use this
option with the --I option.

--l label
Retrieves a custom dialog descriptor installed on the
given instance. The instance’s object path must be in
the form type@label or /distinguished[/type:]label... .
The first form finds the label in the name registry. The
second specifies the instance’s location in the
collection hierarchy. The first form can be used to
reference instances in remote TMRs, since instances
registered with the name registry in one TMR are
known to other TMRs. If there is no per-instance customization, wgetdialog exits with an error.

--r resource
Specifies the type of resource. This option is not used
when getting a dialog descriptor from an instance.
When --r is used by itself (no --c, --C, or --I), the
wgetdialog

wgetdialog looks for a custom version of the dialog for the resource. If there is a custom version, wgetdialog returns it. Otherwise, wgetdialog returns the original version of the dialog.

-s Retrieves the original, stock version of the dialog even if there is a customization installed.

-t tmr Retrieves the dialog descriptor from the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where wgetdialog is run.

-x Retrieves the icon dialog descriptor for the given resource from the transition presentation object.

dialog-name Name of the dialog to retrieve.

EXAMPLES

This example retrieves the add_dialog dialog from the ManagedNode resource.

% wgetdialog -r ManagedNode add_dialog > add.d

SEE ALSO

wlsdialog, wputdialog, wrmdialog
NAME

wgetpending

PURPOSE

Lists the actions associated with a profile.

SYNOPSIS

wgetpending profile

DESCRIPTION

The wgetpending command lists the actions that are scheduled to run the next time the specified profile is distributed to subscribers.

Authorization

super

Arguments

label The object path of a resource instance. The path must be in the form @type:label or /distinguished[/type:]label... . The first form finds the label in the TME 10 name registry. The second specifies the instance's location in the collection hierarchy.

EXAMPLES

The example below gets the current state of the managed node moab. The command looks for the resource name in the name registry.

% wgetstate @ManagedNode:moab
internet_gate

SEE ALSO

wlsicons, wputicon, wputstate, wrmicon
wgetstate

NAME

wgetstate

PURPOSE

Returns the current icon state of a given instance.

SYNOPSIS

wgetstate –l resource

DESCRIPTION

The wgetstate command returns the current icon state of a resource instance.

Authorization

super

Arguments

–a Filters for post-distribution actions that execute on the host where the profile resides.
–A Filters for post-distribution actions that execute on each endpoint.
–b Filters for pre-distribution actions that execute on the host where the profile resides.
–B Filters for pre-distribution actions that execute on each endpoint.
–c Filters for actions associated with record creation. Cannot be used if –d or –m is used.
–d Filters for actions associated with record deletion. Cannot be used if –c or –m is used.
–m Filters for actions associated with modification of records. Cannot be used if –c or –d is used.
EXAMPLES

The example below prints all the actions defined for the engineers profile.

```bash
% wlsactions @UserProfile:engineers
add_entry
assign_office
change_entry
remove_entry
```

The example below prints all actions associated with the creation of profile records.

```bash
% wlsactions -c @UserProfile:engineers
add_entry
assign_office
```

SEE ALSO

waddaction, wgetaction, wrmaction
NAME

wlsactions

PURPOSE

Lists the actions associated with a profile.

SYNOPSIS

wlsactions [-a | -b | -A | -B] [-c | -d | -m] profile

DESCRIPTION

The wlsactions command lists the distribution actions associated with a profile. The command only lists actions that were installed with waddaction.

The command’s options control the types of actions that are listed. The -a, -b, -A, and -B options specify whether the action runs after or before distribution and where the action executes. If you do not specify either option, wlsactions prints all varieties actions. You can only specify one of these options.

The -c, -d, and -m options can be used to filter for actions associated with creation, deletion, or modification of profile records, respectively. If you do not use any of the options, wlsactions does not take into account the action’s event.

wlsactions prints the list of actions to standard output, one action per line.

Authorization

super

Arguments

- Filters for post-distribution actions that execute on the host where the profile resides.
- A Filters for post-distribution actions that execute on each endpoint.
- b Filters for pre-distribution actions that execute on the host where the profile resides.
Filters for pre-distribution actions that execute on each endpoint.

Filters for actions associated with record creation. Cannot be used if \(-d\) or \(-m\) is used.

Filters for actions associated with record deletion. Cannot be used if \(-c\) or \(-m\) is used.

Filters for actions associated with modification of records. Cannot be used if \(-c\) or \(-d\) is used.

**EXAMPLES**

The example below prints all the actions defined for the `engineers` profile.

```
% wlsactions @UserProfile:engineers
add_entry
assign_office
change_entry
remove_entry
```

The example below prints all actions associated with the creation of profile records.

```
% wlsactions -c @UserProfile:engineers
add_entry
assign_office
```

**SEE ALSO**

`waddaction`, `wgetaction`, `wrmaction`
NAME

wlsbitmap

PURPOSE

Prints a list of bitmaps installed on a resource.

SYNOPSIS

 wlsbitmap [-C] -r resource [-t tmr]

DESCRIPTION

The wlsbitmap command provides a list of bitmaps on a resource type. The command also lists the bitmaps for which there are resource-wide customizations. It does not show bitmaps that have been customized on individual instances.

The output of wlsbitmap depends on which language environment (locale) you are using. The output generated in the C locale has the following format:

 bitmap1 (customization status)
 bitmap2 (customization status)
 ...

Each line contains the name of a bitmap. If there is a custom version of the bitmap, wlsbitmap notes it in the customization status field.

Authorization

super

Arguments

-C Generates a list of bitmaps used by all create dialogs of a resource type.

-r resource Specifies the type of resource.

-t tmr Gets the bitmap list for the resource in the TMR specified by the tmr parameter. The named TMR must be connected to the TMR where wlsbitmap is run.
EXAMPLES

% wlsbitmap -r TaskLibrary
bitmap name (customization status)

  task_library
  task (resource-wide customization)
  job

SEE ALSO

  wconbitmap, wgetbitmap, wputbitmap, wrmbitmap
NAME

wlsdialog

PURPOSE

Prints a list of dialog descriptors installed on a resource.

SYNOPSIS

wlsdialog [-C] -r resource [-t tmr]

DESCRIPTION

The wlsdialog command provides a list of dialog descriptors on a resource type. The command also lists the dialog descriptors for which there are resource-wide customizations. It does not show dialogs that have been customized on individual instances.

The output of wlsdialog depends on which language environment (locale) you are using. The output generated in the C locale has the following format:

dialog1 (customization status)
dialog2 (customization status)
...

Each line contains the name of a dialog. If there is a custom version of the dialog, wlsdialog notes it in the customization status field.

Authorization

super

Arguments

-C Generates a list of create dialogs for a resource type.
-r resource Specifies the type of resource.
-t tmr Gets the dialog list for the resource in the TMR specified by the tmr parameter. The named TMR must be connected to the TMR where wlsdialog is run.
EXAMPLES

% wlsdialog -r TaskLibrary
dialog name (customization status)

task_output
task_list
job_list
view_dialog
execute_task (resource-wide customization)
create_task
create_job
save_host_and_path (resource-wide customization)
host_and_path

SEE ALSO

wgetdialog, wputdialog, wrmdialog
NAME

wlsicon

PURPOSE

Lists the icon state defined for a resource type.

SYNOPSIS

wlsicon \(-r\) \(resource\) \([-t\) \(tmr\) \][-x\]

DESCRIPTION

The \texttt{wlsicon} command lists all the icon states defined for a given resource type.

Authorization

\texttt{super}

Arguments

\texttt{\(-r\) \(resource\)} Specifies the type of resource.

\texttt{\(-t\) \(tmr\)} Lists the states on the resource in the TMR named by the \texttt{tmr} parameter. The named TMR must be connected to the TMR where \texttt{wlsicon} is run.

\texttt{\(-x\)} Instructs the command to list the icons installed on the TME 10 transition presentation object.

EXAMPLES

\[
\% wlsicon \ -r\ ManagedNode \ -t\ Wasatch
\]

sunos4-client

sunos4-server

solaris2-client

solaris2-server

hpux9-client

hpux9-server

sysv4-m88k-client

sysv4-m88k-server

aix3-r2-client

aix3-r2-server

dgux5-client

dgux5-server

sysv4-att-client

sysv4-att-server

uw2-ix86-client
uw2-ix86-server
nextstep3-ix86-client
nextstep3-ix86-server
aix4-r1-client
aix4-r1-server
hpux10-client
hpux10-server
w32-ix86-server
w32-ix86-client
mips-irix5-server
mips-irix5-client
osf-axp-server
osf-axp-client
internet_gate

Note: The nextstep3-ix86, mips-irix5, and osf-axp are not currently available in TME 10 AEF 3.0. Contact your support provider for information on the availability of these platforms.

SEE ALSO

wputicon, wputstate, wrmicon
NAME

wputbitmap

PURPOSE

Installs a custom bitmap.

SYNOPSIS

wputbitmap –r resource [-t tmr... -T] bitmap < bitmap.xpm
wputbitmap –l label bitmap < bitmap.xpm
wputbitmap {–I state | –x} –r resource [-t tmr... | -T] < bitmap.xpm

DESCRIPTION

The wputbitmap command installs a custom bitmap. You can use wputbitmap to install customizations of existing bitmaps or to add your own original bitmaps. When you install a customization for an existing bitmap, the original is not destroyed. The customization simply overrides the original.

The wputbitmap command allows you to customize bitmaps for an entire resource type, instances of a resource, and create dialogs. If the bitmap is used by a create dialog, you can install it on all defaults policies or on individual policies. wputbitmap can also install custom icon bitmaps.

The TME Desktop uses bitmaps and icons that are stored in the X pixmap (XPM) format. The XPM format is used extensively in X Windows, since it allows an image to be displayed on monochrome, grey scale, or color displays.

If you want to use your favorite bitmap editor and it does not support XPM, you can still create TME bitmaps and icons. See the wconbitmap reference page for a procedure to convert bitmaps into the format required by TME 10.

If you are customizing an icon bitmap on an application that uses the TME 10 transition presentation object, use the –x option. A pre-2.0 application running under TME 2.0 does not store its icon bitmap and dialog on its presentation objects. Instead, the application stores them on a presentation object provided by TME 10. This presentation
object, called the transition presentation object, is shared by all “transition” applications installed in the TMR. Refer to the application documentation to determine how the application you are modifying stores its icon bitmaps and dialogs.

`wputbitmap` reads the bitmap file from standard input. You must redirect an XPM-format bitmap to the command’s standard input.

**Authorization**

super

**Arguments**

`-I state`  Installs the icon bitmap for the given resource state. The state must already exist. To create a new resource state, use the `wputstate` command. You cannot use this command with the `-x` option.

`-x`  Installs the icon bitmap on the TME 10 transition presentation object. You cannot use this command with the `-I` option.

`-l label`  Installs a custom bitmap on the given instance. The instance’s object path must be in the form `@type:label` or `/distinguished[type]:label...`. The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy. The first form can be used to reference instances in remote TMRs, since instances registered with the name registry in one TMR are known to other TMRs.

`-r resource`  Specifies the type of resource. This option is not used when installing a bitmap on an instance.

`-t tmr...`  Installs the bitmap on the resource in the TMR named in the `tmr` parameter. The named TMR must be connected to the TMR where `wputbitmap` is run. You can use the `-t` option more than once to install the bitmap in several TMRs. If you use the `-t` option, the bitmap is not installed in the current TMR.

`-T`  Installs a custom bitmap on the named resource in the current TMR and all connected TMRs.
wputbitmap

`bitmap` Name of the bitmap to install. This is the name used to reference the bitmap from within TME 10.

**EXAMPLES**

This example retrieves the `add_dialog` dialog from the `ManagedNode` resource.

**EXAMPLES**

This example installs the `up_arrow` bitmap on the managed node `moab`. The bitmap is read from the file `custom_up.xpm`.

```
% wputbitmap -l @ManagedNode:moab up_arrow < custom_up.xpm
```

**SEE ALSO**

`wconbitmap`, `wgetbitmap`, `wlsbitmap`, `wputstate`, `wrmbitmap`
NAME

wputdialog

PURPOSE

Installs a custom dialog.

SYNOPSIS

wputdialog –r resource [-t tmr... –T] dialog < dialog.d
wputdialog –l label dialog < dialog.d
wputdialog {–C | –c policy} –r resource [-t tmr... | –T] dialog <
dialog.d
wputdialog {–I state | –x} –r resource [-t tmr... | –T] < dialog.d

DESCRIPTION

The wputdialog command installs a custom version of a dialog. When
you install a custom dialog, the original is not destroyed. The
customization simply overrides the original.

The wputdialog command allows you to customize dialogs for entire
resource types, instances of a resource, and create dialogs. If the dialog
is a create dialog, you can customize it on all defaults policies or on
individual policies. wputbitmap can also install custom icon dialog
descriptors.

You can also use wputdialog to associate a new dialog descriptor with
an icon state. To do this, use the –I option and specify the icon state. If
the application uses the TME 10 transition presentation object instead
of providing its own, use the –x option instead of the –I option. Pre-2.0
applications running under TME 2.0 do not store their icons and
bitmaps on a presentation object. Instead, they store them on a
transition presentation object provided by the TMR. Refer to the
application documentation to determine how each application stores
its icons and bitmaps.

wputbitmap reads the dialog descriptor from standard input. You
must redirect an compiled descriptor to the command’s standard input.
wputdialog

**Authorization**

*super*

**Arguments**

`–C` Installs a custom create dialog for use by all defaults policies of the given resource type.

`–c policy` Installs a custom create dialog on a specific defaults policy.

`–I state` Installs a custom create dialog on a specific defaults policy.

`–l label` Installs a custom dialog on the given instance. The instance’s object path must be in the form @type:label or /distinguished[type]:label... . The first form finds the label in the name registry. The second specifes the instance’s location in the collection hierarchy. The first form can be used to reference instances in remote TMRs, since instances registered with the name registry in one TMR are known to other TMRs.

`–r resource` Specifies the type of resource. This option is not used when installing a dialog on an instance.

`–t tmr...` Installs the dialog on the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where `wputdialog` is run. You can use the `–t` option more than once to install the dialog in several TMRs. If you use the `–t` option, the dialog is not installed in the current TMR.

`–T` Installs a custom dialog on the named resource in the current TMR and all connected TMRs.

*dialog* Name of the dialog to install. This is the name used to reference the dialog from within TME 10.
EXAMPLES

This example installs a custom create dialog on the BasicTaskLibrary default policy of the TaskLibrary resource.

% wputdialog -c BasicTaskLibrary -r TaskLibrary \   BasicTaskLibrary < make_tl.d

SEE ALSO

wgetdialog, wlsdialog, wputstate, wrmdialog
NAME

wputicon

PURPOSE

Defines a new resource state and installs an icon for the state.

SYNOPSIS

wputicon -r resource [-t tmr...] [-T] { state | -x } dialog-file
bitmap-file

DESCRIPTION

The wputicon command defines a new state for a resource and associates a custom icon with the new state. The icon consists of a bitmap and an icon dialog descriptor.

The dialog-file parameter is the path to the icon’s dialog descriptor, and bitmap-file is the path to the icon’s bitmap. The command makes copies of these files, so you can remove them after you install the icon.

The state parameter must be the name of a new state. If there is already a state with the given name, wputicon exits with an error.

If you are installing a new icon for an application that uses the TME 10 transition presentation object, use the -x option instead of supplying a state name. A pre-2.0 application running under TME 2.0 does not store its icon bitmap and dialog on its presentation object. Instead, the application stores them on a presentation object provided by TME 10. This presentation object, called the transition presentation object, is shared by all “transition” applications installed in the TMR. Refer to the application documentation to determine how the application you are working with stores its icon bitmaps and dialogs.

Installing a custom icon does not cause instances of the resource to begin using it. You must use wputstate to change the state of specific instances.

Authorization

super
Arguments

- `–r resource` Specifies the type of resource.
- `–t tmr...` Installs the icon on the resource in the TMR named in the `tmr` parameter. The named TMR must be connected to the TMR where `wputicon` is run. You can use the `–t` option more than once to install the icon in several TMRs. If you use the `–t` option, the icon is not installed in the current TMR.
- `–T` Installs a custom icon on the named resource in the current TMR and all connected TMRs.
- `state` The name of the new state.
- `–x` Instructs the command to use the TME 10 transition presentation object instead of the resource’s own presentation object.
- `dialog-name` The name of the file that contains the icon’s dialog descriptor.
- `bitmap-name` The name of the file that contains the icon’s XPM bitmap.

EXAMPLES

This example defines a new state called `internet_gateway` on the `ManagedNode` resource type. The new state’s icon is defined by the `gateway.d` dialog descriptor and the `gateway.xpm` bitmap.

```
% wputicon –r ManagedNode internet_gate gateway.d \
gateway.xpm
```

SEE ALSO

`wconbitmap`, `wlsicon`, `wputstate`, `wrmicon`
wputmeth

NAME

wputmeth

PURPOSE

Installs a custom method on a resource or on instances of a resource.

SYNOPSIS

wputmeth [-i interp] [-a acl]... [-u user -g group] [-t tmr...] -r resource...
  method path
wputmeth [-i interp] [-a acl]... [-u user -g group] -l label... method path
wputmeth [-i interp] [-a acl]... [-u user -g group] [-t tmr...] -T -c policy
  -r resource... method path

DESCRIPTION

When users request an operation from a dialog gadget, the TME 10 desktop performs an operation known as a callback method. A callback method is the means by which dialogs and gadgets respond to user input. Callbacks typically transfer data and control from the user interface to underlying application resources.

The wputmeth command installs a custom callback method. You can use wputmeth to install a custom method on a resource-wide basis or on individual instances of a resource type.

Dialogs used to create instances of a resource must exist independently of the instances they create. Because of this, the callback methods associated with a create dialog are stored with the resource’s defaults policies. wputmeth can install custom methods for a create dialog.

The command’s path parameter is the path to the method script. The path might look absolute (it begins with a '/' character), but it is actually a relative path. The TME interprets the path relative to the install_dir/bin directory. So, if the method script is located in install_dir/bin/sunos4/CUSTOM, the path parameter should be /sunos4/CUSTOM/method.sh.
Authorization

super

Arguments

- `a acl...` Specifies an authorization role for the method. You can use `a` more than once to specify several roles. When a method is invoked, TME 10 checks the invoker’s authorization roles against the method’s supported roles. The `acl` parameter must be a valid authorization role defined in TME 10. If you do not use the `a` option, the method can be run by a user with any role.

- `c policy` Installs the method on a defaults policy. This has the effect of installing a custom method associated with a create dialog.

- `g group` Sets the effective group name that the method uses when it is invoked. You must use the `u` option with this option.

- `i interp` Installs a platform-specific version of the method. The `interp` parameter must be one of the following: `aix3-r2`, `aix4-r1`, `default`, `dgux`, `hpux9`, `hpux10`, `solaris2`, `sunos4`, `sysv4-m88k`, or `sysv4-att`. If you do not specify the `i` option, `wputmeth` assumes the interpreter type is `default`.

- `l label...` The object paths of one or more resource instances. Each path must be in the form `type@label` or `distinguished[type:label]...`. The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy.

- `r resource...` Specifies the type of resource you want to customize. This option is not used when installing a method on an instance.

- `t tmr...` Installs the method on the resource in the TMR named in the `tmr` parameter. The named TMR must be connected to the TMR where `wputmeth` is run. You
wputmeth

- can use \texttt{-t} more than once to install the method in several TMRs.

- \texttt{T} Installs the method on the resource in the current TMR and all connected TMRs.

- \texttt{u user} Sets the effective user name that the method uses when it is invoked. You must use the \texttt{-g} option with this option.

- \texttt{method} The name of the custom method.

- \texttt{path} The path to the method script.

\textbf{EXAMPLES}

The example below installs the HP-UX version of a method on the \texttt{ManagedNode} resource type. The method is assigned \texttt{super} and \texttt{senior} as authorization roles.

\begin{verbatim}
  wputmeth -i hpux9 -a super -a senior -r ManagedNode
  set_location /hpux9/CUSTOM/set_location.sh
\end{verbatim}

\textbf{SEE ALSO}

\texttt{wrmmeth}
NAME

wpstate

PURPOSE

Changes the state of an instance.

SYNOPSIS

wpstate –l label... state

DESCRIPTION

The wpstate command sets the state of an instance. When you change the state of an instance, the TME 10 Desktop begins using the icon that is associated with the state.

Authorization

super

Arguments

–l label  Installs a custom bitmap on the given instance. The instance’s object path must be in the form @type:label or /distinguished[type:]label... . The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy. The first form can be used to reference instances in remote TMRs, since instances registered with the name registry in one TMR are known to other TMRs.

state  Specifies the new state for the given instances.

EXAMPLES

The example below sets the state of the managed node moab.

% wpstate @ManagedNode:moab internet_gate

SEE ALSO

wconbitmap, wlsicons, wputicon, wrmicon
NAME

wrmaction

PURPOSE

Removes a custom action script from a profile.

SYNOPSIS

wrmaction \{-a | -b | -A | -B\} \{-c | -d | -m\} profile action

DESCRIPTION

The `wrmaction` command removes an action script from a profile. The action must have been installed with `waddaction`.

You must specify whether the action script runs before or after distribution and whether the script executes on the host where the profile resides or on each endpoint. You must use either the `-a`, `-b`, `-A`, or `-B` option.

You must also specify the type of event associated with the action (creation, deletion, or modification of profile records). Use one of the following options to indicate the type of event: `-c`, `-d`, or `-m`.

Authorization

super

Arguments

- `-a` Specifies that the script runs on the host where the profile resides after the profile is distributed to its subscribers.
- `-A` Specifies that the script runs on each endpoint after the profile is distributed to its subscribers.
- `-b` Specifies that the script runs before the profile is distributed to its subscribers.
- `-B` Specifies that the script runs on each endpoint before the profile is distributed to its subscribers.
- `-c` Specifies that the action is associated with the creation of profile records. Cannot be used if `-d` or `-m` is used.
wrmaction

-d Specifies that the action is associated with the deletion of profile records. Cannot be used if -c or -m is used.

-m Specifies that the action is associated with the modification of profile records. Cannot be used if -c or -d is used.

EXAMPLES

The example below removes the assign_office action from the engineers profile. The action runs after distribution and is associated with record creation.

```bash
% wrmaction -a -c engineers assign_office
```

SEE ALSO

waddaction, wgetaction, wlsactions
NAME
wrmatr

PURPOSE
Removes a custom value from instances of a resource.

SYNOPSIS
wrmatr attribute label...

DESCRIPTION
The wrmatr command removes a custom attribute from one or more instances of a resource. You can only remove attributes that were created with waddattr.

Authorization
super

Arguments
attribute The name of the attribute to remove.
label The object paths of one or more resource instances. Each path must be in the form type@label or /distinguished[type:]label... . The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy.

EXAMPLES
The following example command removes the location attribute from an instance of the ManagedNode resource type.

% wrmatr location @ManagedNode:moab

SEE ALSO
waddatr, getattr, wsetattr
NAME

wrmbitmap

PURPOSE

Removes a custom bitmap from a resource or instance.

SYNOPSIS

wrmbitmap [-C] –r resource [-t tmr... | –T] bitmap
wrmbitmap –l label bitmap

DESCRIPTION

The wrmbitmap command removes a custom bitmap from an instance or resource type. You can only remove bitmaps that were installed with wputbitmap. wrmbitmap allows you to remove custom bitmaps from an entire resource type, instances of a resource, and create dialogs. If the bitmap is associated with a create dialog, you can remove the bitmap from all defaults policies or from individual policies.

You can only remove a bitmap in the same context as it was installed. For example, if you installed the bitmap as a resource-wide customization, you must remove it on that basis. You cannot install a resource-wide bitmap and then remove it from a few instances.

Authorization

super

Arguments

–C

Removes a custom bitmap that was installed on the defaults policies of the given resource types. Custom bitmaps associated with application create dialogs are stored with the resource’s defaults policies. The bitmap must have been installed with wputbitmap.

–l label

Removes a custom bitmap installed on the given instance. The instance’s object path must be in the form @type:label or /distinguished[/type:]label... The first form finds the label in the name registry. The
wrmbitmap

second specifies the instance’s location in the collection hierarchy. The first form can be used to reference instances in remote TMRs, since instances registered with the name registry in one TMR are known to other TMRs.

–r resource Specifies the type of resource. This option is not used when removing a bitmap from an instance.

–t tmr... Removes the bitmap from the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where wrmbitmap is run. You can use the –t option more than once to remove the bitmap from several TMRs. If you use the –t option, the bitmap is not removed from the current TMR.

–T Removes a custom bitmap from the current TMR and all connected TMRs.

bitmap Name of the bitmap to remove. You can use the wlsbitmap command to get a list of a resource’s bitmaps.

EXAMPLES

This example removes the left_arrow bitmap from the TaskLibrary resource.

% wrmbitmap -r TaskLibrary left_arrow

This example removes the smiley bitmap from the task library tools.

% wrmbitmap -l @TaskLibrary:tools smiley

SEE ALSO

wconbitmap, wgetbitmap, wlsbitmap, wputbitmap
NAME

wrmdialog

PURPOSE

Removes a custom dialog from a resource or instance.

SYNOPSIS

wrmdialog -r resource [-t tmr... | -T] dialog
wrmdialog -l label dialog
wrmdialog {–C | –c policy} -r resource [-t tmr... | -T] dialog

DESCRIPTION

The wrmdialog command removes a custom dialog from an instance or resource type. You can only remove dialogs that were installed with wputdialog. wrmdialog allows you to remove custom dialogs from an entire resource type, instances of a resource, and create dialogs. If the dialog is a custom version of a create dialog, you can remove it from all defaults policies or from individual policies.

You can only remove a dialog in the same context as it was installed. For example, if you installed the as a resource-wide customization, you must remove it on that basis. You cannot install a resource-wide dialog and then remove it from a few instances.

Authorization

super

Arguments

–C

Removes a custom create dialog from that was installed across all defaults policies of the resource.

–c policy

Removes a custom create dialog from a specific defaults policy.

–l label

Removes a custom dialog installed on the given instance. The instance’s object path must be in the form @type:label or /distinguished[type:label]... . The first form finds the label in the name registry. The second specifies the instance’s location in the
wrmdialog

The first form can be used to reference instances in remote TMRs, since instances registered with the name registry in one TMR are known to other TMRs.

- \textit{r resource} Specifies the type of resource. This option is not used when removing a dialog from an instance.

- \textit{t tmr...} Removes the dialog from the resource in the TMR named in the \textit{tmr} parameter. The named TMR must be connected to the TMR where \textit{wrmdialog} is run. You can use the \textit{–t} option more than once to remove the dialog from several TMRs. If you use the \textit{–t} option, the dialog is not removed from the current TMR.

- \textit{T} Removes a custom dialog from the current TMR and all connected TMRs.

- \textit{dialog} Name of the dialog to remove. You can use the \textit{wlsdialog} command to get a list of a resource’s dialogs.

**EXAMPLES**

This example removes the custom version of the \textit{tmr_toles} dialog from the \textit{Administrator} resource.

```
% wrmdialog -r Administrator tmr_roles
```

**SEE ALSO**

\textit{wgetdialog, wlsdialog, wputdialog}
NAME

wrmicon

PURPOSE

Removes a custom icon and resource state.

SYNOPSIS

wrmicon -r resource { state | –x } [-t tmr... | -T]

DESCRIPTION

The **wrmicon** command removes a custom icon and its resource state. The icon must have been installed with *wputicon*.

If you are removing an icon from an application that uses the TME 10 transition presentation object, use the –x option instead of supplying a state name. A pre-2.0 application running under TME 2.0 does not store its icon bitmap and dialog on its presentation object. Instead, the application stores them on a presentation object provided by TME 10. This presentation object, called the transition presentation object, is shared by all “transition” applications installed in the TMR. Refer to the application documentation to determine how the application you are working with stores its icon bitmaps and dialogs.

Authorization

super

Arguments

- **-r resource** Specifies the type of resource.
- **-t tmr...** Removes the icon on the resource in the TMR named in the *tmr* parameter. The named TMR must be connected to the TMR where *wputicon* is run. You can use the –t option more than once to remove the icon in several TMRs. If you use the –t option, the icon is not removed from the current TMR.
- **-T** Removes the icon from the named resource in the current TMR and all connected TMRs.
- **state** The name of the resource state to remove.
wrmicon

–x  Instructs the command to look for the application’s icon on the TME 10 transition presentation object instead of on the resource’s own presentation object.

EXAMPLES

This example removes the internet_gate icon from the ManagedNode resource.

% wrmicon –r ManagedNode internet_gate

SEE ALSO

wlsicon, wputicon
NAME

wrmmeth

PURPOSE

Installs a custom method on a resource or on instances of a resource.

SYNOPSIS

wrmmeth [-i interp] [-t tmr... | -T] -r resource... method
wrmmeth [-i interp] -l label... method
wrmmeth [-i interp] [-t tmr... | -T] -c policy -r resource... method

DESCRIPTION

The wrmmeth command removes a custom method that was installed with wputmeth. You can remove custom methods from an entire resource type, from individual instances, or from create dialogs.

Dialogs used to create instances of a resource must exist independently of the instances they create. Because of this, the callback methods associated with a create dialog are stored with the resource’s defaults policies.

Authorization

super

Arguments

–i interp

Removes a platform-specific version of the method. The interp parameter must be one of the following:
aix3-r2, aix4-r1, default, dgux, hpux9, hpux10, solaris2, sunos4, sysv4-m88k, or sysv4-att. You must use the same interpreter value used to install the method with wputmeth. If the method was installed without the –i option, do not use it with wrmmeth.

–c policy

Removes the method from a defaults policy. This has the effect of deleting a custom method associated with a create dialog.

–l label...

The object paths of one or more resource instances. Each path must be in the form type@label or
wrmmeth

/distinguished[/type:]label..., The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy.

–r resource... Specifies the type of resource that contains the method to remove. This option is not used when removing a method from an instance. You can specify this option multiple times to remove a method from several resource types.

–t tmr... Removes the method on the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where wrmmeth is run. You can use –t more than once to remove the method in several TMRs.

–T Removes the method on the resource in the current TMR and all connected TMRs.

method The name of the custom method.

EXAMPLES

The example below removes the HP-UX version of a method on the ManagedNode resource type.

% wrmmeth -i hpux9 -r ManagedNode set_location

SEE ALSO

wputmeth
NAME
wrmpolm

PURPOSE
Removes a custom policy method from a resource type.

SYNOPSIS
wrmpolm [–t tmr... | –T] [–d | –v] resource method

DESCRIPTION
The waddpolm command removes a policy method from all policies of a resource type. You can only remove policy methods that were installed with waddpolm.

Authorization
super

Arguments
–d Indicates that the policy method is a default policy.
–t tmr... Removes the method from the resource in the TMR named in the tmr parameter. The named TMR must be connected to the TMR where wrmpolm is run. You can use –t more than once to install the method in several TMRs.
–T Removes the method from the resource in the current TMR and all connected TMRs.
–v Indicates that the policy method is a validation policy.
resource The resource type from which to remove the method.
method The name of the policy method to remove.

EXAMPLES
The example below removes the nm_val_reboot validation method from the ManagedNode resource type.

% wrmpolm -T -v ManagedNode mn_val_reboot
SEE ALSO

waddpolm, wrunpolm
NAME
  wrmprop

PURPOSE
  Removes a custom property from a profile.

SYNOPSIS
  wrmprop profile property

DESCRIPTION
  The wrmprop command removes a custom property from a profile. The command can only remove properties that were added with waddprop.

Authorization
  super

Arguments
  profile  The name of the profile from which to remove the property.
  property The name of the property to remove.

EXAMPLES
  The example above removes the extension property from the engineers profile.

% wrmprop @UserProfile:engineers extension

SEE ALSO
  waddprop
NAME

wrunpolm

PURPOSE

Runs a custom policy method from within a callback method.

SYNOPSIS

wrunpolm [-i] [-p region] method args...

DESCRIPTION

The wrunpolm command runs a custom policy method from within a callback method. Policy methods are usually invoked internally from within one or more application methods. However, you must manually run custom policy methods.

The wrunpolm command invokes the policy method on the behalf of the policy region where the object resides. To invoke the method on behalf of another region, use the -p option.

Authorization

super

Arguments

-i Causes the command’s standard input to be passed to the policy method’s standard input.

-p region Validates the object against validation policy in a given policy region. If you do not use the -p option, wrunpolm runs the policy method in the object’s home policy region.

method Name of the custom policy method to run.

args... List of arguments to be passed to the policy method.

EXAMPLES

The example below checks the invoking object against the mn_val_reboot policy in the Dallas policy region.

wrunpolm -p /Regions/PolicyRegion:Dallas mn_val_reboot
SEE ALSO

waddpolm, wrmpolm
NAME

wself

PURPOSE

Returns the information about the object that invoked the current callback.

SYNOPSIS

wself [–l]

DESCRIPTION

With no arguments, wself returns the object reference of the resource instance that invoked the current callback. If you invoke wself outside a callback, the command returns OBJECT_NIL.

If you give the –l option, the command prints the object’s label instead of the object reference. If you run wself with the –l option outside of a callback, the command prints an error message.

Authorization

super

Arguments

–l Forces the command to print the object’s label instead of the object reference.
NAME

wsetattr

PURPOSE

Sets the value of a custom attribute on one or more instances of a resource.

SYNOPSIS

wsetattr value attribute label...

DESCRIPTION

The wsetattr command sets the value of a custom attribute that was created with waddattr.

Authorization

super

Arguments

-v value The new value of the attribute.
attribute The name of the attribute to set.
label... The object paths of one or more resource instances. Each path must be in the form type@label or /distinguished[type]:label... . The first form finds the label in the name registry. The second specifies the instance’s location in the collection hierarchy.

EXAMPLES

The following example command sets the value of the location attribute on an instance of the ManagedNode resource type.

% wsetattr "Chicago Office" location @ManagedNode:moab

SEE ALSO

waddattr,wgetattr,wrmattr
NAME

    wstatusline

PURPOSE

Displays a message in a desktop’s status window.

SYNOPSIS

    wstatusline message

DESCRIPTION

The wstatusline command displays a formatted message in the status window of a TME 10 desktop. The message displays on the desktop where the callback method that invoked wstatusline is running.

Authorization

    super

Arguments

    message        The text of the message that is to appear in the status window. The text must be pre-formatted.

EXAMPLES

    % wstatusline "Successfully removed the selected users."

SEE ALSO

    wdispconf, wdisperr, wdispmsg
C

DSL Gadget/Attribute Reference
NAME

Accelerator

PURPOSE

Specifies the keyboard accelerators for the current DSL gadget.

SYNTAX

Accelerator = value

DESCRIPTION

The Accelerator attribute controls the keyboard accelerators for the current DSL gadget. An accelerator is a key sequence that provides a shortcut to selecting a gadget. The Accelerator attribute is supported by the Button, Menu, and Switch gadgets.

The Accelerator gadget’s value must be a string that defines an accelerator key sequence. To specify a control character, use "CTRL+<char>" for the value.

SEE ALSO

Button, Menu, Switch
NAME

Activate

PURPOSE

DSL attribute that specifies a callback list that is invoked when a table row or column is activated.

SYNOPSIS

Activate = callback1, callback2...

DESCRIPTION

The Activate attribute specifies a callback list to be invoked for a row or column of a Table gadget. When you define activation callbacks, a button appears at the beginning of the row or column. If the user presses the button, the activation callbacks are invoked.

The attribute’s value is a comma-separated list of callback methods. The default value is an empty callback list.

SEE ALSO

TableColumn, TableRow
Alignment

NAME

Alignment

PURPOSE

DSL attribute that specifies the alignment of dialog or gadget elements.

SYNOPSIS

Alignment = value

DESCRIPTION

The Alignment attribute controls the alignment of gadgets in a dialog or group, or the alignment of choices in a choice list. This attribute is valid for dialogs and for the following gadgets: Choice, Group, List, and Message.

As an attribute of a dialog, Alignment affects only the top-level gadgets. It does not affect gadgets inside Group gadgets.

As an attribute of a Group gadget, Alignment affects only the top-level gadgets. It does not affect the contents of Group gadgets inside the current Group gadget.

As an attribute of a Choice or List gadget, Alignment affects the alignment of the list items. The attribute only has an effect if the gadget’s Show attribute is set to SOME.

The Alignment attribute’s value must be one of the following keywords:

TOP Aligns the gadgets horizontally along the top of the dialog or group. This value is not valid for Choice, List, or Message gadgets.

BOTTOM Aligns the gadgets horizontally along the bottom of the dialog or group. This value is not valid for Choice, List, or Message gadgets.

LEFT Aligns the gadgets or list choices vertically along the left-hand side of the dialog or gadget.
### Alignment

| **RIGHT** | Aligns the gadgets or list choices vertically along the right-hand side of the dialog or gadget. |
| **CENTER** | Horizontally centers the gadgets or list choices in the dialog or gadget. |

**SEE ALSO**

Choice, Group, List, Message
AppendText

NAME

AppendText

PURPOSE

DSL attribute that specifies text to be appended to Page gadget.

SYNOPSIS

AppendText = string

DESCRIPTION

The AppendText attribute appends a given string to the end of a Page gadget. This attribute is only valid for Page gadgets.

The attribute’s value is a string of arbitrary length. You can append a multiple lines of text by embedding newline (\n) characters in the string.

EXAMPLES

Page {
    Name = results;
    Title = "Policy Results";
    Value = $PR_RESULTS;
    AppendText = $DATE;
}

SEE ALSO

Page
NAME

Background

PURPOSE

DSL attribute that specifies the background color of a dialog or gadget.

SYNTAX

Background = name | value

DESCRIPTION

The Background attribute controls the background color of the current DSL gadget or dialog.

The attribute’s value can be expressed two ways:

name  A color name. See the dsl_colors reference page for a complete list of color names.

value  A hexadecimal color value in the form: #rrggbb.

The color value is represents a color in the RGB color model. The RGB color model specifies three components for each color: red, green, and blue. The rr variable specifies the red component, the gg specifies the green component, and bb specifies the blue component. The hexadecimal values specify the component intensity: 00 represents off and ff represents full on. So, Background = #00ff00 specifies a green background, because the green component is turned full on and the red and blue components are turned off. All shades of all colors can be produced by changing the intensity of one or more color components. White is produced by turning all three components full on, and black is produced by turning all three components off.

Mixing the color components to obtain a specific color shade using hexadecimal values can be complex. You should use this technique only when the required color is not available by name. See your X users guide for more details about color specification and the RGB color model.

SEE ALSO

dsl_colors, Foreground
**Bitmap**

**NAME**

Bitmap

**PURPOSE**

DSL attribute that specifies the name of the bitmap used to display an icon in DSL.

**SYNTAX**

`Bitmap = bitmap-name`

**DESCRIPTION**

The Bitmap attribute specifies the name of the bitmap to be used to display an icon. This attribute is valid only in Icon gadgets.

You can specify the value of BitmapTitle in two ways. The first way is to specify the absolute path to a bitmap file. The second way is to give the name of a bitmap stored on the same resource as the dialog. The default value of the attribute is the value of the Type attribute.

**Example**

```plaintext
Icon {
    Name     = printer_icon;
    Type     = Printer;
    Bitmap   = prn_icon;
    Commands = prn_icon_menu;
    Open     = show_queue@{...};
}
```

**SEE ALSO**

Icon
NAME

BitmapTitle

PURPOSE

DSL attribute that specifies a bitmap to be used as the title of a dialog or gadget.

SYNOPSIS

BitmapTitle = value

DESCRIPTION

The BitmapTitle attribute specifies a bitmap to be used as the title of a dialog or gadget. This attribute is valid for dialogs and for all gadgets except Separator.

You can specify the value of BitmapTitle in two ways. The first way is to specify the absolute path to a bitmap file. The second way is to give the name of a bitmap stored on the same resource as the dialog.

EXAMPLES

CommandDialog {
    Variables {
        Cstring machine_type;
    }

    Attributes {
        Name = man_node_dialog;
        BitmapTitle = $machine_type;
    }
}
Border

NAME

Border

PURPOSE

DSL attribute that specifies whether to draw a border around a gadget.

SYNTAX

Border = bitmap-name

DESCRIPTION

The Border attribute specifies whether to draw a border around a gadget or group of gadgets. This attribute is valid for the following gadgets: Choice, Collection, Group, List, Message, Page, Switch, Table, and Text.

The Border attribute’s value must be one of the following keywords:

YES Draws a border around the gadget.
NO Suppresses drawing of the border.

The default value is NO.

EXAMPLES

Group {
  Attributes {
    Name = g1;
    Border = YES;
    Layout = VERTICAL;
  }
}

SEE ALSO

Choice, Collection, Group, List, Message, Page, Switch, Table, Text
NAME

Button

PURPOSE

DSL gadget that allows a user to invoke an operation.

SYNTAX

Button {
    Name = gadget-name;
    Title = title;
    Commands = callback1, callback2, ...;
    ...
}

DESCRIPTION

A Button gadget allows a user to invoke an operation. A button can be used by itself or as part of a Menu gadget. When defined on its own, the Button gadget appears as a single button on the display. When defined as part of a Menu gadget, the Button gadget appears as an item in the menu.

In addition to the attributes common to all gadgets, you can define the following attributes for a Button gadget:

Accelerator  Specifies an accelerator key for the button.
Commands     Specifies the callbacks to be invoked when the button is pushed.
Mnemonic     Specifies a mnemonic for the button.

EXAMPLES

Button {
    Name = edit_map_button;
    Title = "Edit Map";
    Commands = edit_maps(...);
}

SEE ALSO

Accelerator, Commands, gadgets, Mnemonic
NAME

CellBorderStyle

PURPOSE

DSL attribute that controls that style of border around cells in a Table gadget.

SYNOPSIS

CellBorderStyle = value

DESCRIPTION

The CellBorderStyle attribute specifies the style of border drawn around the cells in a Table gadget. The attribute’s value must be one of the following keywords:

DASHED_BOX  Draws a dashed box around each cell.
INDENTED_BOX Draws a single line box around each cell.
PLAIN  Inhibits boxes around the cells.

The default value is Plain.

SEE ALSO

Table
NAME

Changed

PURPOSE

DSL attribute that specifies callbacks to be invoked when the user has changed a gadget’s value.

SYNOPSIS

Changed = callback1, callback2...

DESCRIPTION

The Changed attribute defines a list of callbacks to be invoked when the user finishes making changes to a gadget’s value. This attribute is valid for the following DSL gadgets: Choice, Collection, List, Switch, Table, and Text.

For Text gadgets, the desktop invokes the callbacks when the user presses <Return>. For other gadgets, the desktop invokes the callbacks whenever the user changes the gadget’s state.

The Changed attribute’s value is a comma-separated list of callbacks. The default value is an empty callback list.

EXAMPLES

Text {
    Name = new_message;
    Title = "New Message: ";
    Changed = update_message@() < $new_message;
}

SEE ALSO

Choice, Collection, List, Switch, Table, Text
NAME

ChildColumnAlignment

PURPOSE

DSL attribute that specifies the horizontal alignment of a gadget.

SYNTAX

ChildColumnAlignment = value

DESCRIPTION

The ChildColumnAlignment attribute controls the horizontal alignment of gadgets in dialogs that use grid alignment instead of manual layout. This attribute is only available if the Layout attribute of the dialog or gadget that contains the gadget is set to GRID.

The ChildColumnAlignment attribute is valid for the following DSL gadgets: Button, Choice, Collection, Group, List, Message, Page, Separator, Switch, Table, Text.

As an attribute of a Group gadget, ChildColumnAlignment affects only the top-level gadgets in the current Group gadget. It does not affect Group gadgets nested within the gadget.

The ChildColumnAlignment attribute’s value must be one of the following upper-case words:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>Aligns the gadgets along the left-hand side of the dialog or group.</td>
</tr>
<tr>
<td>CENTER</td>
<td>Centers the gadgets horizontally in the dialog or group.</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Aligns the gadgets along the right-hand side of the dialog or group.</td>
</tr>
<tr>
<td>STRETCH</td>
<td>Aligns the gadgets so that they fill the horizontal space allocated for the dialog or group.</td>
</tr>
</tbody>
</table>

This attribute’s default value is CENTER.
 EXAMPLES

Group {
    Attributes {
        Name = g0;
        Layout = GRID;
        ChildColumnAlignment = LEFT;
    }
}

SEE ALSO

Button, ChildRowAlignment, Choice, Collection, Group, Layout, List, Message, Page, Separator, Switch, Table, Text
ChildRowAlignment

NAME

ChildRowAlignment

PURPOSE

DSL attribute that specifies the vertical alignment of a gadget.

SYNTAX

ChildRowAlignment = value

DESCRIPTION

The ChildRowAlignment attribute controls the vertical alignment of gadgets in dialogs that use grid alignment instead of manual layout. This attribute is only available if the Layout attribute of the dialog or gadget that contains the gadget is set to GRID.

The ChildRowAlignment attribute is valid for the following DSL gadgets: Button, Choice, Collection, Group, List, Message, Page, Separator, Switch, Table, Text.

As an attribute of a Group gadget, ChildRowAlignment affects only the top-level gadgets in the current Group gadget. It does not affect Group gadgets nested within the gadget.

The ChildRowAlignment attribute’s value must be one of the following upper-case words:

- TOP: Aligns the gadgets along the top of the dialog or group.
- BOTTOM: Aligns the gadgets along the bottom of the dialog or group.
- CENTER: Aligns the gadgets so they are vertically centered in the dialog or group.
- STRETCH: Aligns the gadgets so that they fill the vertical space allocated for the dialog or group.

This attribute’s default value is CENTER.
EXAMPLES

Group {
    Attributes {
        Name = g0;
        Layout = GRID;
        ChildRowAlignment = TOP;
    }
}

SEE ALSO

Button, ChildColumnAlignment, Choice, Collection, Group, Layout, List, Message, Page, Separator, Switch, Table, Text
Choice

NAME

Choice

PURPOSE

DSL gadget that allows the user to choose exactly one member from a set of values.

SYNTAX

Choice {
    Name = gadget-name;
    Title = title;
    Choices = choice-list;
    Value = initial-choice;
    ...
}

DESCRIPTION

A Choice gadget allows the user to choose exactly one member from a set of values. The gadget can appear as a scrolling list, a row of buttons, or as a single-value indicator. Each choice value is represented by a string displayed to the user.

In addition to the attributes common to all gadgets, you can specify the following attributes for a Choice gadget:

- **Alignment**: Controls the alignment of the values in the gadget.
- **Border**: Specifies the type of border drawn around the gadget.
- **Changed**: Specifies callbacks to be invoked when the user changes the currently selected value.
- **Choices**: Specifies the list of values to be displayed. You must define a value for this attribute.
- **Columns**: Defines the width of the gadget.
- **DoubleClick**: Specifies the callbacks invoked when a user double-clicks on a value.
- **Layout**: Specifies the layout of the choices.
**Choice**

- **ReadOnly**  Controls whether the gadget can be changed by the user.

- **Rows**  Specifies the height of the gadget.

- **Show**  Specifies how many of the choices are displayed. This attribute controls whether the gadget appears as a scrolling list, a row of buttons, or a single-value indicator.

- **ShowBrowser**  Controls whether a choice browser is displayed with the gadget. When selected, the browser displays a dialog of all the items in the list. The user can select list items from the browser dialog as they would from the list.

- **Sort**  Specifies whether to display the choices in sorted order.

- **TellModify**  Specifies how an unchanged value is to be passed to callback methods.

- **Value**  Contains the value that is initially selected when the user is presented with the gadget.

**EXAMPLES**

```
Choice {
    Name = known_nodes;
    Title = "Known Nodes:"
    TitlePos = TOP;
    Choices = $KNOWN_NODES;
}
```

**SEE ALSO**

Alignment, Border, Changed, Choices, CommandDialog, DoubleClick, gadgets, Layout, ReadOnly, Rows, Show, ShowBrowser, Sort, TellModify, Value
NAME

Choices

PURPOSE

DSL attribute that defines the choices for a list.

SYNOPSIS

Choices = value

DESCRIPTION

The Choices attribute defines the list of choices that are displayed in a Choice or List gadget.

The value of the attribute is a comma-separated list of menu items. The format is shown below.

item1{"value1"},item2{"value2"},...,itemn{"valuen"}

The item is the text of the item as it appears in the list. The value of the string is the value of the Value attribute if the user selects the item.

EXAMPLES

Choice {
    Name = shell;
    Title = "Default shell";
    Show = ONE;
    Choices =
        /bin/sh{"SH"},/bin/csh{"CSH"},/bin/tcsh{"TCSH"};
}

SEE ALSO

Choice, List
NAME

Collection

PURPOSE

DSL gadget that presents a dialog sector containing a collection of Icon gadgets.

SYNOPSIS

Collection {
    Name = gadget-name;
    Title = gadget-title;
    CopyIn = drag rules for copying icons into the gadget;
    MoveIn = drag rules for moving icons into the gadget;
    MoveOut = drag rules for moving icons out of the gadget;
    ...
}

DESCRIPTION

The Collection gadget is used to display a set of icons. Its primary use is as a view into the contents of an object. For example, the TaskLibrary resource’s view dialog is a Collection gadget that contains icons for all the task and job objects in the library.

In addition to the attribute available to all DSL gadgets, you can set the following attributes on a Collection gadget:

Border Controls whether a border is drawn around the gadget.

Changed Specifies a list of callbacks that are invoked whenever the value of the gadget changes. A Collection gadget’s value changes whenever an icon is added or removed.

CopyIn Defines rules that govern types of objects can be copied into the gadget.

MoveIn Defines rules that govern what types of objects can be moved into the gadget.

MoveOut Defines rules that govern what types of objects can be moved out of the gadget.
**ReadOnly**  Controls whether the user can affect the contents of the gadget.

**TellModify**  Specifies how the gadget’s value is passed to callback methods if the user did not change the value.

**Type**  Associates a type name with the gadget. TME 10 uses the type name when enforcing drag rules.

**Value**  Specifies which icons (if any) are initially selected when the user is presented with the dialog.

**ViewBy**  Controls how icons in the gadget are displayed; by icon bitmap or by label.

**SEE ALSO**

Border, Changed, CopyIn, gadgets, MoveIn, MoveOut, ReadOnly, TellModify, Type, Value, ViewBy
NAME

ColumnSelection

PURPOSE

DSL attribute that specifies how many columns in a Table gadget can be selected at once.

SYNTAX

ColumnSelection = value

DESCRIPTION

The ColumnSelection attribute controls how many columns in a Table gadget can be selected at one time. This attribute is only available if the table’s SelectionMode attribute is set to SELECT_COLUMN or SELECT_BOTH.

Value

The ColumnSelection gadget’s value must be one of the following keywords:

SINGLE_SELECT
   Allows one column at a time to be selected.

MULTIPLE_SELECT
   Allows selection of more than one column at a time.

EXTENDED_SELECT
   Allows selection of a contiguous group of columns.

SEE ALSO

Table, SelectionMode
ColumnWidth

NAME

ColumnWidth

PURPOSE

DSL attribute that specifies the width of a column in a Table gadget.

SYNTAX

ColumnWidth = column_width

DESCRIPTION

The ColumnWidth attribute sets the width of a column in a Table gadget. This attribute is valid for Table, TableRow and TableColumn gadgets. In a Table gadget, the attribute sets the width of all columns. In a TableColumn gadget, the attribute sets the width of that column. In a TableRow gadget, the attribute sets the default width of all cells in the row.

The ColumnWidth attribute’s value must be an integer number of text columns.

SEE ALSO

Columns, Table, TableColumn, TableRow
NAME

Columns

PURPOSE

DSL attribute that specifies the number of text columns in a gadget.

SYNTAX

Columns = gadget_width

DESCRIPTION

The Columns attribute specifies the width of a gadget that displays text. The attribute’s value is the minimum number of columns of text that are displayed. If the gadget This gadget is valid for the following gadgets: Choice, List, Menu, Page, Table, and Text.

In Page and Text gadgets, the attribute sets the width of the gadget. In Choice, List, and Menu gadgets, however, the attribute sets the minimum width of the gadget. The width of the gadget is the value of Columns or the width of the widest list member, whichever is larger.

In a Table gadget, the attribute specifies the number of table columns to display at one time.

The Columns attribute’s value must resolve to a decimal integer. The default value is 20 for Text gadgets and 80 for Page gadgets. The default for other list gadgets is the width of the widest list element. The default for a Table gadget is to show all columns.

EXAMPLES

Page {
    Name = motd;
    Title = "Message of the Day:";
    TitlePos = TOP;
    Rows = 6;
    Columns = 81;
    Value = $MN_MOTD;
}

SEE ALSO

Choice, List, Menu, Page, Rows, Table, Text
NAME

CommandButton

PURPOSE

DSL gadget that allows a user to invoke an operation.

SYNTAX

CommandButton {
    Name = gadget-name;
    Title = title;
    Commands = callback1, callback2, ...;
    ...
}

DESCRIPTION

A CommandButton gadget allows a user to invoke an operation. The command buttons for a dialog always appear along the bottom of the dialog. For example, a dialog’s Apply and Cancel buttons are CommandButton gadgets.

In addition to the attributes common to all gadgets, you can define the following attribute for a CommandButton gadget:

Commands  Specifies the callbacks to be invoked when the button is activated.

EXAMPLES

CommandButton {
    Name = apply;
    Title = "Apply";
    Commands = make_changes(...), dtc_dismiss(...);
}

SEE ALSO

Commands, gadgets
NAME

Command Dialog

PURPOSE

Introduction to DSL command dialogs.

SYNTAX

Command Dialog {
  Variables {
    type variable1;
    type variable2;
    ...
  }
  Attributes {
    Name = dialog-name;
    ...
  }
  Gadgets {
    Gadget definitions
  }
}

DESCRIPTION

Command dialogs allow users to enter values and perform application commands. A Command dialog’s definition is composed of three parts: a Variables block, a Attributes block and a Gadgets block.

The Variables block defines any variables used by the dialog. Variables are often used to store the values of callback methods.

The Attributes block contains attributes that control the dialog’s behavior and appearance. You can specify the following attributes for a Command dialog:

- **Alignment**: Controls the alignment of top-level gadgets in a Command dialog.
- **Background**: Specifies the background color of the dialog.
- **BitmapTitle**: Specifies a bitmap to be used as the dialog title.
Command Dialog

| **Default** | Specifies the default command button in the dialog. |
| **DialogStyle** | Specifies the style of the dialog. |
| **Font** | Specifies the default font for the dialog’s text. |
| **Foreground** | Specifies the default foreground color for the dialog. |
| **HelpMenu** | Defines a help menu for the dialog. |
| **HelpMessage** | Defines a help message displayed at the bottom of the dialog. |
| **Iconic** | Controls whether the dialog initially appears as an icon. |
| **Layout** | Defines the default layout of the dialog’s gadgets. |
| **Name** | Defines the name of the dialog. |
| **PopDown** | Specifies the callbacks invoked when the dialog is removed from a display. |
| **Purpose** | Specifies the dialog’s purpose. |
| **Refresh** | Specifies callbacks for refreshing the dialog. |
| **StatusField** | Specifies whether or not the Command dialog contains an area for displaying status messages. |
| **StatusMessage** | Defines a default status message for the dialog. |
| **Title** | Specifies the title of the dialog. The title is displayed at the top of the dialog. |
| **Visible** | Controls whether the dialog is initially visible to the user. |

The **Gadgets** block defines the gadgets that appear in the dialog. You can define the following gadgets in a dialog: **Button**, **Choice**, **Collection**, **CommandButton**, **Group**, **Icon**, **List**, **Menu**, **Message**, **Page**, **Separator**, **Switch**, **Table**, **TableColumn**, **TableRow**, and **Text**.

**EXAMPLES**

```c
Command Dialog {
    Variables {
        CString CURRENT_SUBSCRIBERS;
    }
```


CString AVAILABLE_SUBSCRIBERS;
}

Attributes {
    Name = select_subscribers;
    Title = "Select Subscribers";
    PopDown = dtc_free($owner, select_subscribers);
}

Gadgets {
    ...
}

SEE ALSO

Alignment, Background, BitmapTitle, Button, Choice, Collection, CommandButton, Default, DialogStyle, Font, Foreground, gadgets, Group, HelpMenu, HelpMessage, Icon, Iconic, Layout, List, Menu, Message, Name, Page, PopDown, Purpose, Refresh, Separator, StatusField, StatusMessage, Switch, Table, TableColumn, TableRow, Text, Title, Visible
Commands

NAME

Commands

PURPOSE

DSL attribute that specifies a list of callbacks that are invoked when a gadget is selected.

SYNTAX

Commands = callback1, callback2...

DESCRIPTION

The Commands attribute is valid for Button, CommandButton, and Icon gadgets.

For Button and CommandButton gadgets, the attribute defines a list of callbacks invoked when the button is selected. The attribute’s value is a comma-separated list of callback names.

For Icon gadgets, the attribute specifies the gadget path name of the Menu gadget that defines the icon’s pop-up menu.

SEE ALSO

Button, CommandButton, Icon
NAME

CopyIn

PURPOSE

DSL attribute that defines the drag rules for copying icons into a collection or onto another icon.

SYNTAX

CopyIn = rule1, rule2, ..., rulen

DESCRIPTION

The CopyIn attribute defines drag rules to be used when icons are copied from one location into a Collection gadget or onto another icon. This attribute is valid only in Collection and Icon gadgets.

The CopyIn attribute’s value is a comma separated list of drag rules. Each drag rule has the following format:

source {subject} [callbacks] message

source Specifies the type of source collection from which icons can be copied. The source collection’s Type attribute must match this type. The source value is a quoted string. If you omit this field, the drag rule does not restrict the types of collections from which the view dialog can receive icons.

subject Specifies the type of icon the destination can receive from the source collection. The subject value is a quoted string. If you omit this field, the drag rule does not restrict the types of icons that can be moved into the view dialog from the source collection. There is only one subject per drag rule. To define a set of icon types for a single source, define multiple drag rules and specify the same source.

callbacks Specifies a command-separated list of callbacks that are invoked after the drag operation is validated. The callback list can contain both method callbacks and desktop commands.
CopyIn

message Specifies a status message that is displayed in the dialog’s status field when the drag rule is invoked. The message is optional.

A drag rule list without a subject defines the default CopyIn callbacks. When an icon of type X is copied into the collection, and no drag rules for type X are defined, the default callbacks are invoked.

If you don’t define a value for the CopyIn attribute, users cannot copy icons into the collection or icon.

EXAMPLES

Collection {
    Name = view_dialog;
    Type = ManagedNode;
    Title = "Full Client: ";

    CopyIn = {ManagedNode} [dropin@ManagedNode] < $drops],
            {TaskLibrary} [dropin@TaskLibrary] < $drops],
            {Task} [dropout@Task] < $drops];
}

SEE ALSO

Collection, Icon, MoveIn
NAME

Default

PURPOSE

DSL attribute that specifies the default command button in a dialog.

SYNTAX

Default = button-name

DESCRIPTION

The Default attribute sets the default command in a dialog. This provides the user with a shortcut to activate the most commonly used button on the dialog. For example, if your dialog has a single dismiss button, setting the dialog’s Default attribute allows the user to close the dialog simply by pressing <Return>.

This attribute is only valid on dialogs.

The Default attribute’s value must be the name of a CommandButton gadget on the dialog. This attribute’s default value is the name of the first command button in the dialog’s Commands attribute.

SEE ALSO

CommandDialog
DialogStyle

NAME

DialogStyle

PURPOSE

DSL attribute that defines the style of a dialog.

SYNTAX

DialogStyle = value

DESCRIPTION

The DialogStyle attribute controls the whether the user can interact with other dialogs while the dialog is active.

DialogStyle is only valid for dialogs.

The DialogStyle attribute’s value must be one of the following keywords:

MODELESS    The user can interact with any other dialog on the Desktop.

SYSTEM_MODAL The user cannot interact with any other dialogs on the Desktop — including non-TME 10 dialogs and windows — until the dialog is closed.

FULL_APPLICATION_MODAL The user cannot interact with any other TME 10 dialogs until the dialog is closed.

PRIMARY_APPLICATION_MODAL The user cannot interact with any parents of the dialog until it is closed.

This attribute’s default value is MODELESS.

SEE ALSO

CommandDialog
NAME

DoubleClick

PURPOSE

Defines actions associated with double-click selection.

SYNTAX

DoubleClick = callback1, callback2...

DESCRIPTION

The DoubleClick attribute defines a list of callbacks that are invoked when a user double clicks on a list element or table cell.

DoubleClick is valid for the following gadgets: Choice, List, Table, TableColumn, and TableRow.

The DoubleClick attribute’s value is a comma-separated list of callbacks.

SEE ALSO

Choice, List, Table, TableColumn, TableRow
Encrypt

NAME

Encrypt

PURPOSE

Specifies whether to encrypt the value of a gadget.

SYNTAX

Encrypt = encryption_type

DESCRIPTION

The Encrypt attribute controls whether the value of Text gadget is encrypted before passing it to a callback method. This attribute is valid only in Text gadgets.

If encryption is enabled for a Text gadget, the Desktop displays that gadget’s contents as a number of hollow boxes equal to the number of characters in the gadget’s value.

The value of the Encrypt attribute must be one of the following keywords:

CRYPT   Encrypts the text with the standard UNIX password-encryption scheme.

DES      Encrypts the text using the Data Encryption Standard. This option is only available on U.S. versions of TME 10.

NONE     Performs no encryption; the text string is sent "in the clear."

The default value of the is NONE.

Note: The result of DES encryption is not a printable string. You must always pass a DES-encrypted string to a callback method on the method’s standard input.

Example

```text
Text {
   Name = password;
   Title = "Password:"
   TellModify = YES;
}
```
Encrypt

Encrypt = CRYPT;
}

SEE ALSO

Text
Font

NAME

Font

PURPOSE

DSL attribute that specifies the font of a gadget.

SYNOPSIS

Font = value

DESCRIPTION

The Font attribute sets the font used to display text in a gadget. This attribute is valid for all DSL gadgets. Setting the attribute at the dialog level sets the default font used by all gadgets in the dialog. The value of the Font attribute must be expressed in XLFD format. The default value of the attribute is the default font of the dialog.

EXAMPLES

Text {
    Name = foo;
    Title = "Foo Field";
    Font =
        -adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1;
}
NAME

Foreground

PURPOSE

DSL attribute that specifies the foreground color of a dialog or gadget.

SYNTAX

Foreground = name | value;

DESCRIPTION

The Foreground attribute controls the background color of the current DSL gadget or dialog.

Value

The attribute’s value can be expressed two ways:

name A color name. See the dsl_colors reference page for a complete list of color names.

value A hexadecimal color value in the form: #rrggbb.

The color value is represents a color in the RGB color model. The RGB color model specifies three components for each color: red, green, and blue. The rr variable specifies the red component, the gg specifies the green component, and bb specifies the blue component. The hexadecimal values specify the component intensity: 00 represents off and ff represents full on. So, Foreground = #00ff00 specifies a green foreground, because the green component is turned full on and the red and blue components are turned off. All shades of all colors can be produced by changing the intensity of one or more color components. White is produced by turning all three components full on, and black is produced by turning all three components off.

Mixing the color components to obtain a specific color shade using hexadecimal values can be complex. You should use this technique only when the required color is not available by name. See your X users guide for more details about color specification and the RGB color model.
SEE ALSO
dsl_colors, Foreground
NAME
 Glyph

PURPOSE
 DSL attribute that associates a small bitmap with a TableColumn or TableRow gadget.

SYNOPSIS
 Glyph = bitmap-name

DESCRIPTION
 The Glyph attribute associates a bitmap with a table row or column. The glyph bitmap appears between the title and the first cell of the row or column. The Glyph attribute is often used to display a pictograph that indicates the state of the row or column. For example, you might use a bitmap of a lock to indicate that a table row is read-only.

This attribute is only valid for TableColumn and TableRow gadgets.

SEE ALSO
 TableColumn, TableRow
GridHeight

NAME

GridHeight

PURPOSE

DSL attribute that defines the height of the grid in a Group or Table gadget.

SYNOPSIS

GridHeight = value

DESCRIPTION

The GridHeight attribute sets the height of a Group or Table gadget that uses grid alignment. The gadget’s Layout attribute must be set to GRID.

The value of the attribute is the number of text rows in the gadget.

SEE ALSO

GridWidth, Group, Layout, Table
GridHorizontal

NAME

GridHorizontal

PURPOSE

Specifies the horizontal placement of a gadget relative to a grid.

SYNOPSIS

GridHorizontal = value

DESCRIPTION

The GridHorizontal attribute specifies the horizontal placement within a grid. This attribute is only available if the Layout attribute of the dialog or gadget that contains the gadget is set to GRID.

This attribute is valid for the following gadgets: Button, Choice, Collection, List, Message, Page, Switch, Table, and Text.

The attribute’s value is a decimal integer that represents the number of text columns the gadget is offset from the origin. The position of the origin depends on the values of the ChildColumnAlignment and ChildRowAlignment attributes.

SEE ALSO

Button, ChildColumnAlignment, ChildRowAlignment, Choice, Collection, GridVertical, List, Message, Page, Switch, Table, Text.
GridVertical

NAME

GridVertical

PURPOSE

Specifies the vertical placement of a gadget relative to a grid.

SYNOPSIS

GridVertical = value

DESCRIPTION

The GridVertical attribute specifies the vertical placement within a grid. This attribute is only available if the Layout attribute of the dialog or gadget that contains the gadget is set to GRID.

This attribute is valid for the following gadgets: Button, Choice, Collection, List, Message, Page, Switch, Table, and Text.

The attribute’s value is a decimal integer that represents the number of text rows the gadget is offset from the origin. The position of the origin depends on the values of the ChildColumnAlignment and ChildRowAlignment attributes.

SEE ALSO

Button, ChildColumnAlignment, ChildRowAlignment, Choice, Collection, GridHorizontal, List, Message, Page, Switch, Table, Text,
GridWidth

NAME

GridWidth

PURPOSE

DSL attribute that defines the width of the grid in a Group gadget.

SYNOPSIS

GridWidth = value

DESCRIPTION

The GridWidth attribute sets the width of a Group gadget that uses grid alignment. The gadget’s Layout attribute must be set to GRID. The value of the attribute is the number of text columns in the gadget.

SEE ALSO

GridHeight, Group, Layout
Group

NAME

Group

PURPOSE

DSL gadget that forms a logical collection of gadgets.

SYNTAX

Group {
  Attributes {
    Name = gadget-name;
    Title = gadget-title;
    ...
  }
  Gadgets{
    gadgets in the group
  }
}

DESCRIPTION

A Group gadget organizes a set of logically related DSL gadgets. TME 10 arranges the group’s members so the user can see that they are related. A Group gadget can contain other Group gadgets, but cannot contain Menu gadgets.

As shown above, the DSL syntax of a Group gadget is different from the syntax of most other gadgets. You must place the Group gadget’s attributes in an Attributes block. The group’s members are defines in a Gadgets block.

In addition to the attributes common to all gadgets, you can specify the following attributes for a Group gadget:

Alignment  Controls the alignment of the top-level gadgets in the group. The alignment does not affect gadgets in nested Group gadgets.

Border      Specifies the type of border drawn around the gadget.

GridHeight  Defines the height of the Group gadget (if the gadget uses grid alignment).
**GridWidth**  Defines the width of the **Group** gadget (if the gadget uses grid alignment).

**Layout**  Specifies the layout of the gadgets contained in the **Group** gadget.

**Scrolling**  Specifies whether to make the **Group** gadget scroll if TME 10 determines that it is too "tall."

**EXAMPLES**

```
Group {
   Attributes{
      Name = g0;
      Layout = HORIZONTAL;
   }
   Gadgets {
      Message {
         Name = md;
         Title = "Policy Region: ";
         Value = $HDO_MDO;
      }
      Text {
         Name = location;
         Title = "Location: ";
      }
   }
}
```

**SEE ALSO**

Alignment, Border, gadgets, GridHeight, GridWidth, Layout, Scrolling
HelpMenu

NAME

HelpMenu

PURPOSE

DSL attribute that specifies the name of a menu to be used as a help menu.

SYNOPSIS

HelpMenu = gadget_path

DESCRIPTION

The HelpMenu attribute names a Menu gadget that is to be used as a dialog’s help menu. The menu selected as the help menu always appears on the right-hand side of the dialog.

This attribute is only valid for dialogs.

The attribute’s value must be the gadget path of a Menu gadget. The default value is null.

SEE ALSO

HelpMessage
HelpMessage

NAME

HelpMessage

PURPOSE

DSL attribute that associates a status message with a gadget.

SYNTAX

HelpMessage = string

DESCRIPTION

The HelpMessage attribute specifies a status message to be displayed when a user points at an icon or menu item. This attribute is valid for dialogs and for the Menu and Button gadgets.

If HelpMessage is specified for a dialog, a CommandButton labeled Help... appears on the dialog. When the user presses this button, a dialog that contains the help text appears. When used with a Menu or Button gadget, the help text appears on the dialog’s status line whenever the menu or button is highlighted. Gadget help text only displays if the StatusField dialog attribute is set to YES.

The value of the HelpMessage is a text string. You can use the Msg directive to retrieve the string from a message catalog. The attribute’s default value is an empty string.

EXAMPLES

Button {
    Name = return;
    Title = "Return";
    HelpMessage = "Return to previous menu."
    Commands = return@($owner, view_dialog);
}

SEE ALSO

HelpMenu, StatusField
Icon

NAME

Icon

PURPOSE

DSL gadget that represents an object on the Desktop.

SYNTAX

Icon {
    Name = gadget-name;
    Bitmap = bitmap-name;
    Commands = name of the icon's pop-up menu;
    Open = callbacks to invoke when the icon is opened;
    Type = icon's type string;
    Object = ID of the object that the icon represents;
    ...
}

DESCRIPTION

An Icon gadget represents a single object on the display. With an icon, a user can perform operations on the object that corresponds to the icon. With an icon, the user can:

- Open the icon, thus displaying another dialog.
- Display a menu of object commands.
- Select the icon as an argument to commands.
- Drag a selected icon and drop it onto another icon or into a Collection gadget.

An icon is displayed using a predefined bitmap image stored in X pixmap format. An Icon gadget's Bitmap attribute determines its default bitmap.

Dropping an icon onto a drop site (another icon or a Collection gadget) changes the relationship between the dropped object and its original location, and establishes a new relationship between the object and its new location. For example, dropping an object icon that represents a printer object onto a Collection gadget that represents a policy region removes the printer object from its previous policy region and adds it...
Icon

to the new policy region. You can specify **Icon** and **Collection** gadgets as drop sites by specifying drag rules with the icon’s **MoveIn** and **CopyIn** attributes.

In addition to the attributes common to all gadgets, you can specify the following attributes for an **Icon** gadget:

- **Bitmap**  
  Specifies the name of the bitmap used to display the icon.

- **Commands**  
  Specifies the name of the **Menu** gadget that defines the icon’s pop-up menu.

- **CopyIn**  
  Specifies rules for copying objects into the icon’s object.

- **MoveIn**  
  Specifies rules for moving objects into the icon’s object.

- **Object**  
  Specifies the object reference of the object that the icon represents. In general, DSL files do not specify a value for this attribute. Instead, its value is set at run-time when populating a **collection** gadget.

- **Open**  
  Defines callbacks to be invoked when the icon is opened.

- **Type**  
  Specifies the type of the icon, which determines what drag-and-drop rules are enforced on the icon.

**EXAMPLES**

```plaintext
Icon {
  Name = prn_icon;
  Type = Printer;
  HelpMessage   = "Full Client";
  Bitmap   = printer;
  Commands = prn_icon_menu;
  Open     = show_queue@();
}
```

**SEE ALSO**

- **Bitmap**, **Commands**, **CopyIn**, **gadgets**, **MoveIn**, **Object**, **Open**, **Type**
Layout

NAME

Layout

PURPOSE

DSL attribute that dictates the layout of a dialog, group, or choice list.

SYNOPSIS

Layout = value

DESCRIPTION

The Layout attribute controls the general layout of a dialog, group, or choice list. The attribute is valid for dialogs and for the following gadgets: Choice, Group, and List.

The attribute’s value must be one of the following keywords:

HORIZONTAL

Sets the general layout of the dialog or gadget to horizontal.

VERTICAL

Sets the general layout of the dialog or gadget to vertical.

GRID

Forces the dialog or gadget to use grid alignment. This allows you to use the grid alignment attributes to position the elements that comprise the dialog or gadget.

SEE ALSO

ChildColumnAlignment, ChildRowAlignment, Choice, GridHeight, GridHorizontal, GridVertical, GridWidth, Group, List
NAME

LayoutPolicy

PURPOSE

DSL attribute that defines whether a Table gadget is populated horizontally or vertically.

SYNOPSIS

LayoutPolicy = value

DESCRIPTION

The LayoutPolicy attribute defines whether a Table gadget is populated by rows or by columns. The attribute’s value must be one of the following keywords:

COLUMN_MAJOR
The table is populated by columns.

ROW_MAJOR
The table is populated by rows.

The attribute’s default value is ROW_MAJOR.

SEE ALSO

Table
LeftColumn

NAME

LeftColumn

PURPOSE

DSL attribute that controls which column in a Table gadget is initially displayed as the left-hand column.

SYNOPSIS

LeftColumn = value

DESCRIPTION

The LeftColumn attribute dictates which column in a scrolling Table gadget is initially displayed as the left-hand column. Any columns to the left of the designated column can be viewed by scrolling to port.

This attribute is only defined for Table attributes.

SEE ALSO

Table
NAME

List

PURPOSE

DSL gadget that allows the user to choose several members from a set of values.

SYNTAX

List {
    Name = gadget-name;
    Title = title;
    Choices = choice-list;
    Value = initial-choice;
    ...
}

DESCRIPTION

A List gadget allows the user to choose several members from a set of values. A List is similar to a Choice gadget, except that Choice gadgets only allows the user to select one value. The gadget can appear as a scrolling list, a row of buttons, or as a single-value indicator. Each element in the list is represented by a string displayed to the user.

In addition to the attributes common to all gadgets, you can specify the following attributes for a List gadget:

Alignment    Controls the alignment of the values in the gadget.
Border       Specifies the type of border drawn around the gadget.
Changed      Specifies callbacks to be invoked when the user changes the currently selected value.
Choices      Specifies the list of values to be displayed. You must define a value for this attribute.
Columns      Defines the width of the gadget.
DoubleClick Specifies the callbacks invoked when a user double-clicks on a value.
Layout       Specifies the layout of the choices.
List

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Controls whether the gadget can be changed by the user.</td>
</tr>
<tr>
<td><strong>Rows</strong></td>
<td>Specifies the height of the gadget.</td>
</tr>
<tr>
<td><strong>Show</strong></td>
<td>Specifies how many of the choices are displayed. This attribute controls whether the gadget appears as a scrolling list, a row of buttons, or a single-value indicator.</td>
</tr>
<tr>
<td><strong>ShowBrowser</strong></td>
<td>Controls whether a choice browser is displayed with the gadget. When selected, the browser displays a dialog of all the items in the list. The user can select list items from the browser dialog as they would from the list.</td>
</tr>
<tr>
<td><strong>Sort</strong></td>
<td>Specifies whether to display the choices in sorted order.</td>
</tr>
<tr>
<td><strong>TellModify</strong></td>
<td>Specifies how the gadget value is passed to callback methods if the user did not change the value.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Contains the values (if any) that are initially selected when the user is presented with the gadget.</td>
</tr>
</tbody>
</table>

**EXAMPLES**

```c
List {
    Name = avail_subscr;
    Title = "Available Subscribers";
    TitlePos = TOP;
    Choices = $KNOWN_NODES;
}
```

**SEE ALSO**

Alignment, Border, Changed, Choices, CommandDialog, DoubleClick, gadgets, Layout, ReadOnly, Rows, Show, ShowBrowser, Sort, TellModify, Value
NAME

Menu

PURPOSE

DSL gadget that allows the user to select from a menu (or submenu) of choices.

SYNTAX

Menu {
  Attributes {
    Name = gadget-name;
    Title = title;
    ...
  }

  Contents {
    Button and Menu gadgets that comprise the menu
  }
}

DESCRIPTION

A menu is a set of selections and submenus that gives users access to a set of operations. A Menu gadget contains Button gadgets that define the menu’s choices. A menu can also contain other Menu gadgets, which allows you to define submenus of a menu.

One of the main uses of Menu gadgets is to define the pop-up menu associated with an Icon gadget. To create an icon pop-up menu, define a Menu gadget and then set the Icon gadget’s Commands attribute to the name of the Menu gadget.

Menus can be shared between dialogs or among many icons by defining them in one dialog and referring to them by name in other dialogs or in icon definitions. Menu sharing considerably reduces the size of dialog descriptors stored on objects and reduces the memory consumed by a dialog displayed on the Desktop. Menu sharing also makes it easy to maintain icon menus, since all icons of the same type share the same menu. However, unlimited sharing means that Menu
gadget names must be globally unique. A simple convention of constructing names by concatenating the type and context of the Menu gadget should avoid most conflicts. For example, printer_icon_menu identifies the printer type in the context of an icon’s pop-up menu.

In addition to the attributes common to all DSL gadgets, Menu gadgets also support the following attributes:

- **Accelerator**: Defines an accelerator key for the gadget.
- **Columns**: Defines the width of the gadget.
- **Mnemonic**: Specifies a mnemonic for the menu name.
- **TearOff**: Controls whether or not the menu can be "torn off" of its source dialog. Tear-off menus can be moved and resized independently of the source dialog.

### EXAMPLES

```plaintext
Menu {
  Attributes {
    Name = printer_queue_menu;
    Title = "Queue";
  }

  Contents {
    Button {
      Name = flush;
      Title = "Flush";
      HelpMessage = "Flush jobs from the queue";
      Commands = flush_queue@(...);
    }

    Button {
      Name = restart;
      Title = "Restart";
      HelpMessage = "Restart the queue";
      Command = restart_queue@(...);
    }
  }
}
```

### SEE ALSO

Accelerator, Collection, Columns, gadgets, Icon, Mnemonic, TearOff
NAME

Message

PURPOSE

DSL gadget that displays a read-only text string.

SYNTAX

Message {
    Name = gadget-name;
    Value = "message-text";
    ...
}

DESCRIPTION

A Message gadget displays a text string. A message is an inactive gadget, which means that the user cannot alter its value. In addition to the attributes common to all gadgets, a Message gadget supports the following attributes:

- **Alignment**
  Sets the horizontal alignment of the text within the gadget.

- **Border**
  Controls the type of border drawn around the gadget.

- **Value**
  Sets the message text to be displayed. The string can contain multiple lines of text separated by newline (\n) characters.

EXAMPLES

```
Message {
    Name = hair_status;
    Title = "Hair Status: ";
    Value = "Your hair is on fire.";
}
```

SEE ALSO

Alignment, Border, gadgets, Value
Mnemonic

NAME

Mnemonic

PURPOSE

DSL attribute that specifies a mnemonic for a menu or menu choice.

SYNOPSIS

Mnemonic = value

DESCRIPTION

The Mnemonic attribute defines a mnemonic for a menu or menu choice. A mnemonic is a letter that the user can press to select a menu or choice. If the mnemonic is a letter in the gadget title, the letter is underlined. The attribute is valid for the following gadgets: Menu, Button, Switch.

The value of the attribute is a single printable character.

SEE ALSO

Button, Menu, Switch
NAME

MoveIn

PURPOSE

DSL attribute that defines the drag rules for moving icons into a collection or onto another icon.

SYNTAX

MoveIn = rule1, rule2, ..., ruleN

DESCRIPTION

The MoveIn attribute defines drag rules to be used when icons are moved from one location and dropped into a Collection gadget or onto another icon. This attribute is valid only in Collection and Icon gadgets.

The MoveIn attribute’s value is a comma separated list of drag rules. Each drag rule has the following format:

source {subject} [callbacks] message

- **source**: Specifies the type of source collection from which icons can be copied. The source collection’s Type attribute must match this type. The source value is a quoted string. If you omit this field, the drag rule does not restrict the types of collections from which the collection can receive icons.

- **subject**: Specifies the type of icon the destination can receive from the source collection. The subject value is a quoted string. If you omit this field, the drag rule does not restrict the types of icons that can be moved into the collection from the source collection. There is only one subject per drag rule. To define a set of icon types for a single source, define multiple drag rules and specify the same source.

- **callbacks**: Specifies a command-separated list of callbacks that are invoked after the drag operation is validated. The
callback list can contain both method callbacks and desktop commands.

message  Specifies a status message that is displayed in the dialog’s status field when the drag rule is invoked. The message is optional.

A drag rule list without a subject defines the default MoveIn callbacks. When an icon of type X is moved into the view, and no drag rules for type X are defined, the default callbacks are invoked.

If you don’t define a value for the MoveIn attribute, users cannot move icons into the collection or icon.

**EXAMPLES**

```plaintext
Collection {
    Name      = view_dialog;
    Type      = ManagedNode;
    Title     = "Full Client: ";

    MoveIn = {ManagedNode} [dropin@(ManagedNode) < $drops],
             {TaskLibrary} [dropin@(TaskLibrary) < $drops],
             {Task} [dropout@{Task} < $drops];
}
```

**SEE ALSO**

Collection, CopyIn, Icon, MoveOut, Type
NAME
MoveOut

PURPOSE
DSL attribute that defines the drag rules for moving icons out of a collection.

SYNTAX
MoveOut = rule1, rule2, ..., rulen

DESCRIPTION
The MoveOut attribute defines drag rules to be used when icons are moved out of a collection. This attribute is only valid in Collection gadgets.

The MoveOut attribute’s value is a comma separated list of drag rules. Each drag rule has the following format:

{subject} [callbacks] message

subject Specifies a type of icon that can be moved out of the collection. The subject value is a quoted string. The desktop compares the value of the icon’s Type attribute against this value when enforcing drag rules. If you omit this field, the drag rule does not restrict the types of icons that can be moved out of the collection.

callbacks Specifies a command-separated list of callbacks that are invoked after the drag operation is validated. The callback list can contain both method callbacks and desktop commands.

message Specifies a status message that is displayed in the dialog’s status field when the drag rule is invoked. The message is optional.

A drag rule list without a type symbol specifies the default MoveOut callbacks. When an icon of type X is moved out and no drag rules for type X are defined, the default callbacks are invoked.

If you don’t define a value for the MoveOut attribute, users cannot move objects out of the collection.
MoveOut

EXAMPLES

Attributes {
  Name      = view_dialog;
  Type      = ManagedNode;
  Title     = "Full Client: ";

  MoveOut = {ManagedNode} [dropout@(ManagedNode) < $drops],
            {TaskLibrary} [dropout@(TaskLibrary) < $drops],
            {Task}[dropout@{Task} < $drops];
}

SEE ALSO

Collection, CopyIn, MoveIn, Type
NAME

Name

PURPOSE

DSL attribute that defines the name of a dialog or gadget.

SYNTAX

Name = value

DESCRIPTION

The **Name** attribute specifies the name of a dialog or gadget. This attribute must be specifically defined in every dialog and gadget. The name of a gadget is used to reference the gadget’s value.

The **Name** attribute’s value may be any sequence of alphanumeric characters that is not a reserved word in DSL. All gadgets in a dialog must have unique names. This attribute has no default value.
Object

NAME
Object

PURPOSE
DSL attribute that specifies the object ID of the object to which an icon corresponds.

SYNTAX
Object = objref

DESCRIPTION
The Object attribute specifies the object ID of the object to which an icon corresponds. This attribute is valid only for Icon gadgets. Define this attribute when populating a Collection gadget. In some cases, method-valued dialog variables can be useful for defining this attribute’s value.

The Object attribute’s value be a valid object reference.

EXAMPLES
Icon {
    Name = client;
    Type = ManagedNode;
    Bitmap = sunos4-client;
    Object = $OID;
    Bitmap = host;
    Open = post_view@();
}

SEE ALSO
Collection, Icon
NAME

Off

PURPOSE

DSL attribute that specifies the value returned when a Switch gadget is in the "off" position.

SYNTAX

Off = value

DESCRIPTION

The Off attribute defines the value of the switch when it is off. This attribute is only valid for Switch gadgets.

The Off attribute’s value can be any string. Its default value is "0".

EXAMPLES

Switch {
    Name = logins_enabled;
    Title = "Logins Enabled:"
    On = "ENABLED";
    Off = "DISABLED";
}

SEE ALSO

On, Switch
On

NAME

On

PURPOSE

DSL attribute that specifies the value returned when a Switch gadget is in the "on" position.

SYNTAX

On = value

DESCRIPTION

The On attribute defines the value of the switch when it is activated. This attribute is only valid for Switch gadgets.

The On attribute’s value can be any string. Its default value is "1".

EXAMPLES

Switch {
    Name = logins_enabled;
    Title = "Logins Enabled:";
    On = "ENABLED";
    Off = "DISABLED";
}

SEE ALSO

Off, Switch
NAME

Open

PURPOSE

DSL attribute that specifies callbacks to be invoked when an icon is opened.

SYNOPSIS

Open = callback1, callback2, ...

DESCRIPTION

The Open attribute defines a list of callbacks that are invoked when an icon is opened. These callbacks typically present another dialog. The callbacks should open the main dialog associated with the icon's resource type.

This attribute is valid for the Icon gadget. The value is a comma-separated list of callbacks. The default value is an empty callback list. If you do not specify a value for the Open attribute, the user cannot open the icon.

EXAMPLES

Icon {
    Name = task_library_icon;
    Type = task_library;
    Bitmap = task_library;
    Commands = tl_menu;
    Open = post_view@();
}

SEE ALSO

Icon
NAME

Page

PURPOSE

DSL gadget that displays one or more lines of text that the user can change.

SYNTAX

Page {
    Name = gadget-name;
    Title = title;
    Value = "Initial text to be edited";
    ...
}

DESCRIPTION

A Page gadget displays one or more lines of text that the user can change. In addition to the attributes common to all gadgets, you can specify the following attributes for a Page gadget:

Border Controls the type of border drawn around the gadget.

Changed Specifies the callbacks invoked when the user edits the text in the gadget.

Columns Specifies the width of the gadget, in columns.

ReadOnly Controls whether the user can change the gadget’s value.

Rows Specifies the height of the gadget.

TellModify Controls how the gadget value is passed to the callback if the user did not change the value.

Value Contains the text that appears in the gadget when the dialog is presented to the user.

EXAMPLES

Page {
    Name = comments;
    Title = "Voice your opinion";
}
Value = "Use this dialog to enter your comments about this software."
Columns = 48;
Rows = 6;
TitlePos = TOP;
}

SEE ALSO

Border, Changed, Columns, gadgets, ReadOnly, Rows, TellModify, Value
PopDown

NAME

PopDown

PURPOSE

DSL attribute that specifies callbacks to be invoked when a dialog is removed from a display.

SYNTAX

PopDown = call_back1, callback2, ...

DESCRIPTION

The PopDown attribute specifies callbacks to be invoked when a dialog is popped down (removed from the display). These callbacks typically perform various kinds of clean-up actions. The PopDown attribute is only valid as a dialog attribute.

The PopDown attribute’s value is a comma-separated list of one or more callbacks.

SEE ALSO

CommandDialog
Purpose

NAME

Purpose

PURPOSE

DSL attribute that defines a dialog’s purpose.

SYNTAX

Purpose = value

DESCRIPTION

The Purpose attribute specifies a dialog’s purpose. The Desktop selects whether the Command dialog is modal depending on this attribute’s value. The Desktop also may display the command dialog differently depending on this attribute’s value.

The Purpose attribute’s value must be one of the following upper-case words:

CONFIRM  The command dialog asks for confirmation before carrying out an operation. This Purpose makes the Command dialog modal.

ERROR   The command dialog displays an error message. This Purpose makes the Command dialog modal.

HELP   The command dialog displays help information. This Purpose may or may not make the Command dialog modal, depending on the characteristics of the local Desktop.

OTHER  The command dialog implements an interaction that does not match any of the other purposes described here. This Purpose does not make the Command dialog modal.

This attribute’s default value is OTHER.

SEE ALSO

CommandDialog, DialogStyle
ReadOnly

NAME

ReadOnly

PURPOSE

DSL attribute that specifies a gadget as read-only.

SYNOPSIS

ReadOnly = value

DESCRIPTION

The ReadOnly attribute specifies a gadget as read-only. The user cannot change the value of a read-only gadget. Setting the ReadOnly attribute is not the same as setting the Sensitive attribute to NO. A read-only gadget appears normal, except that the user cannot change its value. An insensitive gadget appears "greyed out."

This attribute is valid for the following gadgets: Choice, Collection, List, Page, Switch, Table, TableColumn, TableRow, and Text.

The ReadOnly attribute’s value must be one of the following keywords:

YES The gadget is read-only.

NO The gadget is not read-only.

The default value of the attribute is NO.

SEE ALSO

Choice, Collection, List, Page, Sensitive, Switch, Table, TableColumn, TableRow, Text
NAME

Refresh

PURPOSE

DSL attribute that specifies callbacks for refreshing a dialog.

SYNTAX

Refresh = callback_list

DESCRIPTION

The Refresh attribute specifies callbacks to be invoked when the dialog is refreshed (updated so that the dialog’s contents reflect reality). The desktop displaying a dialog and the objects represented by the dialog may be on different parts of a large distributed network. Thus a dialog on a desktop may display out-of-date information that does not reflect the current state of the underlying objects. To handle updating the dialog, use this attribute to specify the callbacks that update the dialog.

The callbacks in this list are invoked whenever the dialog is presented. Desktop commands returned by the callbacks are performed immediately, before the dialog is presented. This behavior allows developers to ensure that the newly displayed dialog contains the most up-to-date information possible.

This attribute is only valid in dialogs. It is not valid in gadgets.

The Refresh attribute’s value is a comma-separated list of callbacks. Its default value is an empty callback list, in which case no callbacks are invoked.

SEE ALSO

CommandDialog
RowHeight

NAME

RowHeight

PURPOSE

DSL attribute that specifies the height of a row in a Table gadget.

SYNTAX

RowHeight = rows

DESCRIPTION

The RowHeight attribute sets the height of a row in a Table gadget. This attribute is valid for Table and TableRow gadgets. In a Table gadget, the attribute sets the default height of all rows. In a TableRow gadget, the attribute sets the height of that row.

The RowHeight attribute’s value must be an integer number of text rows.

SEE ALSO

Rows, Table, TableRow
NAME
RowSelection

PURPOSE
DSL attribute that specifies how many rows in a Table gadget can be selected at once.

SYNTAX
RowSelection = value

DESCRIPTION
The RowSelection attribute controls how many rows in a Table gadget can be selected at one time. This attribute is only available if the table’s SelectionMode attribute is set to SELECT_ROW or SELECT_BOTH.

The RowSelection gadget’s value must be one of the following keywords:

SINGLE_SELECT
  Allows one row at a time to be selected.

MULTIPLE_SELECT
  Allows selection of more than one row at a time.

EXTENDED_SELECT
  Allows selection of a contiguous group of rows.

SEE ALSO
SelectionMode, Table
Rows

NAME

Rows

PURPOSE

DSL attribute that specifies the number of rows in a gadget.

SYNTAX

Rows = gadget_height

DESCRIPTION

The Rows attribute specifies the height of a gadget that displays text. This gadget is valid for the following gadgets: Choice, List, Page, and Table.

In a Table gadget, the attribute specifies the number of table rows to display at one time. In other gadgets, the attribute specifies the number of text rows in the gadget.

The Rows attribute’s value must resolve to a decimal integer.

EXAMPLES

Page {
    Name = motd;
    Title = "Message of the Day:"
    TitlePos = TOP;
    Rows = 6;
    Columns = 81;
    Value = $MN_MOTD;
}

SEE ALSO

Choice, Columns, List, Menu, Page, Table, Text
NAME

Scrolling

PURPOSE

DSL attribute that determines whether a gadget’s members can be presented in a scrolling window.

SYNOPSIS

Scrolling = value

DESCRIPTION

The Scrolling attribute controls whether a group of gadgets can be displayed in a scrolling window. If this attribute is set, TME 10 can display the scrolling window if it determines that a Group or TableColumn gadget is too tall for all its members to be displayed at once. If a TableRow gadget is too wide, TME 10 can display it as a horizontally scrolling window.

This attribute is valid for Group gadgets and for TableRow and TableColumn gadgets.

The Scrolling attribute’s value must be one of the following keywords:

YES Allows the gadget to be displayed as a scrolling gadget if it is necessary.

NO Forces the gadget to be displayed as a whole.

SEE ALSO

Group, TableColumn, TableRow
SelectionMode

NAME

SelectionMode

PURPOSE

DSL attribute that controls what can be selected in a table.

SYNOPSIS

SelectionMode = value

DESCRIPTION

The SelectionMode attribute controls what a user can select in a Table gadget. The attribute is not valid for other types of gadgets.

With the SelectionMode attribute, you can allow a user to select both rows and columns, rows only, columns only, or neither.

The attribute's value must be one of the following keywords:

SELECT_ROW
  Allows the user to select rows only.

SELECT_COLUMN
  Allows the user to select columns only.

SELECT_BOTH
  Allows the user to select either rows or columns.

SELECT_NONE
  Prevents the user from selecting rows or columns. This effectively creates a read-only Table gadget.

SEE ALSO

Table
NAME

Sensitive

PURPOSE

DSL attribute that control whether a user can alter the value of a gadget.

SYNOPSIS

Sensitive = value

DESCRIPTION

The Sensitive attribute controls whether a gadget is sensitive to user input. When a gadget is sensitive, a user can alter its value. The type of window manager the user uses determines how an insensitive gadget appears. For example, in Motif, an insensitive gadgets appears indistinct, also known as "greyed out."

This attribute is valid for all DSL gadgets.

Value

The Sensitive attribute’s value must be one of the following keywords:

YES The gadget is sensitive to user input.
NO The gadget is not sensitive to user input.

The default value of the attribute is YES.

SEE ALSO

ReadOnly
Separator

NAME

Separator

PURPOSE

DSL gadget that places a horizontal or vertical separator in a dialog.

SYNOPSIS

Separator {
  Name = gadget-name;
  ...
}

DESCRIPTION

The Separator gadget inserts a horizontal or vertical separating line in a dialog. The Separator gadget does not have a value and cannot be affected by the user; it is simply a device for spacing gadgets in a dialog.

The gadget supports a subset of the attributes available to all DSL gadgets: Background, ChildColumnAlignment, ChildRowAlignment, Foreground, GridHorizontal, GridVertical, Name, Sensitive, and Visible.

SEE ALSO

gadgets
NAME

Show

PURPOSE

DSL attribute that controls the number of items displayed in a list.

SYNOPSIS

Show = value

DESCRIPTION

The Show attribute specifies the number of items that are shown at once in a Choice or List gadget.

Value

The attribute’s value must be one of the following keywords:

ALL Displays all of the choices as a set of buttons. This value is usually appropriate only for a small set of choices.

SOME Displays as the choices in a scrolling window. The size of the scrolling window is determined by the layout of the gadget. This value should be used if there are more than a few items in the list.

ONE Displays only the currently selected choice. Selecting the gadget causes the other items in the list to display. This value is only valid for Choice gadgets.

NONE Behaves like ONE, except the currently selected choice is not displayed. Instead, an arrow bitmap is displayed. Selecting the arrow bitmap shows the list choices. This value is only valid for Choice gadgets.

SEE ALSO

Choice, List
ShowBrowser

NAME
ShowBrowser

PURPOSE
DSL attribute that specifies whether to offer a browser for a choice list.

SYNOPSIS
ShowBrowser = value

DESCRIPTION
The ShowBrowser attribute controls whether a browser is available for a list. A browser is another way to view a list of items. When selected, the browser displays a dialog of all the items in the list. The user can select list items from the browser dialog as they would from the list.

This attribute is only valid for Choice and List.

The value of the ShowBrowser attribute must be one of the following keywords:

YES A browser is available for the gadget.
NO A browser is not available for the gadget.

The default value of the attribute is YES.

SEE ALSO
Choice, List
ShowIndicator

NAME
ShowIndicator

PURPOSE
DSL attribute that controls whether a switch graphically indicates its state.

SYNOPSIS
ShowIndicator = value

DESCRIPTION
The ShowIndicator attribute controls whether a switch presents an indicator to show when it is on and off.

This attribute is only valid for the Switch gadget.

Value
The value of the ShowIndicator attribute must be one of the following keywords:

YES  The Switch gadget graphically indicates whether it is on or off.

NO   The Switch gadget does not indicate whether it is on or off.

The attribute’s default value is YES.

SEE ALSO
Switch
Sort

NAME
Sort

PURPOSE
DSL attribute that controls whether items in a list are sorted.

SYNOPSIS
Sort = value

DESCRIPTION
The Sort attribute controls whether choices in a choice list are displayed in ascending alphabetical order.
This attribute is only valid for Choice and List gadgets.

Value
The Sort attribute’s value must be one of the following keywords:
YES The choices in the list are sorted.
NO The choices in the list are not sorted.
The default value of the attribute is YES.

SEE ALSO
Choice, List
NAME

StatusField

PURPOSE

DSL attribute that controls whether a dialog has a status message field.

SYNTAX

StatusField = value

DESCRIPTION

The StatusField attribute controls whether a dialog allocates space at the bottom for a status message field. The status message is set with the StatusMessage attribute.

This attribute is only valid at the dialog level.

Value

The StatusField attribute’s value must be one of the following keywords:

YES Allocates space for a status message field.
NO Does not allocate space for a status message field.

The StatusField attribute’s default value is YES.

SEE ALSO

CommandDialog, StatusMessage
StatusMessage

NAME

StatusMessage

PURPOSE

DSL attribute that defines a status message displayed at the bottom of a dialog.

SYNOPSIS

StatusMessage = string

DESCRIPTION

The StatusMessage attribute displays a formatted status message at the bottom of a dialog. This attribute is only valid at the dialog level. The value of the attribute must be a properly quoted string. The default value is an empty string.

If you set the StatusMessage attribute, you must also set the StatusField attribute to YES.

SEE ALSO

CommandDialog, StatusField
Switch

NAME

Switch

PURPOSE

DSL gadget that presents the user with a simple two-state switch.

SYNTAX

Switch {
    Name = gadget-name;
    Title = title;
    On = on-value;
    Off = off-value;
    ...
}

DESCRIPTION

A Switch gadget allows the user to change the on/off state of a switch. Both of the switch states — on and off — can be associated with an arbitrary value. In addition to the attributes common to all gadgets, you can specify the following attributes for a Switch gadget:

Accelerator  Specifies an accelerator key for the button.
Border       Controls the drawing of a border around the gadget.
Changed      Specifies callbacks to be invoked when the user changes the state of the switch.
Mnemonic     Specifies a mnemonic for the button.
Off          Defines the value of the gadget when the switch is turned off.
On           Defines the value of the gadget when the switch is turned on.
ReadOnly     Allows you to prevent the user from changing the state of the switch.
TellModify   Specifies how the gadget value is passed to callback methods if the user does not change the state.
Switch

Value  Specifies the initial state of the switch. The attribute’s value must be either the value of the On attribute or the Off attribute.

EXAMPLES

Switch {
    Name = create_directories;
    Title = "Create directories if missing: ";
    On = "YES";
    Off = "NO";
    Value = "NO";
}

SEE ALSO

Accelerator, Border, Changed, gadgets, Mnemonic, Off, On, ReadOnly, TellModify, Value
NAME

Table

PURPOSE

DSL gadget that presents a multi-row, multi-column table of data that can be edited by a user.

SYNOPSIS

Table {
    Attributes {
        Name = gadget-name;
        Title = title;
        ...
    }

    Contents {
        Optional TableRow and TableColumn gadgets
    }
}

DESCRIPTION

The Table gadget defines a multi-row, multi-column table of data that can be edited by the user.

The Contents block of a Table gadget contains definitions for the rows and/or columns in the table. You must supply definitions for the fixed portion of the table. For example, if your table is five columns wide and an indeterminate number of row high, the Table gadget’s Contents block must contain five TableColumn gadgets. If the table is not fixed in either direction, you can omit the Contents block.

In addition to the attributes common to all gadgets, you can specify the following attributes for a Group gadget:

Border Specifies whether to draw a border around the gadget.

CellBorderStyle Dictates the style of border drawn around each cell in the table.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Changed</strong></td>
<td>Defines a list of callbacks that are invoked when a user changes the value of the gadget.</td>
</tr>
<tr>
<td><strong>ColumnSelection</strong></td>
<td>Controls how many table columns can be selected at once.</td>
</tr>
<tr>
<td><strong>ColumnWidth</strong></td>
<td>Specifies the default width of the table’s columns (can be overridden in a TableColumn gadget).</td>
</tr>
<tr>
<td><strong>Columns</strong></td>
<td>Specifies the number of table columns to display at once.</td>
</tr>
<tr>
<td><strong>DoubleClick</strong></td>
<td>Defines a list of callbacks that are invoked when a user double-clicks on a cell.</td>
</tr>
<tr>
<td><strong>GridHeight</strong></td>
<td>Specifies the grid height of the gadget, if grid alignment is turned on.</td>
</tr>
<tr>
<td><strong>GridWidth</strong></td>
<td>Specifies the grid width of the gadget, if grid alignment is turned on.</td>
</tr>
<tr>
<td><strong>LayoutPolicy</strong></td>
<td>Controls how the table is populated (by row or by column).</td>
</tr>
<tr>
<td><strong>LeftColumn</strong></td>
<td>Specifies which column initially appears as the left-most column in a scrolling table.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Allows the user to browse the contents of the table, but not alter them.</td>
</tr>
<tr>
<td><strong>RowHeight</strong></td>
<td>Specifies the default height of the table’s rows (can be overridden in a TableRow gadget).</td>
</tr>
<tr>
<td><strong>RowSelection</strong></td>
<td>Controls how many table rows can be selected at once.</td>
</tr>
<tr>
<td><strong>Rows</strong></td>
<td>Specifies the number of table rows that are displayed at once.</td>
</tr>
<tr>
<td><strong>SelectionMode</strong></td>
<td>Controls whether the user can select rows only, columns only, or both.</td>
</tr>
<tr>
<td><strong>TellModify</strong></td>
<td>Controls how the gadget’s value is passed to callback methods if the user did not alter the value.</td>
</tr>
</tbody>
</table>
**TopRow**
Specifies which table row initially appears as the top row in a scrolling table.

**Value**
Specifies which row or column is initially selected when the user is presented with the dialog.

**SEE ALSO**
Changed, ColumnSelection, ColumnWidth, Columns, DoubleClick, gadgets, GridHeight, GridWidth, LayoutPolicy, LeftColumn, ReadOnly, RowHeight, RowSelection, Rows, SelectionMode, TellModify, TopRow, Value
**NAME**

TableColumn

**PURPOSE**

DSL gadget that defines a column in a **Table** gadget.

**SYNOPSIS**

```plaintext
Table {
  ...
  Contents {
    TableColumn {
      Name = gadget-name;
      Title = title;
      ...
    }
  }
}
```

**DESCRIPTION**

The **TableColumn** gadget defines a column within a **Table** gadget. You must include one **TableColumn** gadget for every fixed column in your table. For example, if your table always has five columns, you must define five **TableColumn** gadgets.

This gadget can only be used as a subordinate of a **Table** gadget.

In addition to the attributes common to all DSL gadgets, you can specify the following attributes for a **TableColumn** gadget:

- **Activate**
  Specifies a callback list that is invoked when the column is "activated" by the user.

- **Alignment**
  Controls the alignment of the text in the column’s cells.

- **ColumnWidth**
  Specifies the width of the column.

- **DoubleClick**
  Defines a list of callbacks that are invoked when a user double-clicks on a cell in the column.

- **Glyph**
  Associates a bitmap with the column, similar to **BitmapTitle**.
ReadOnly

Allows the user to browse the information in the column, but not alter it.

RowHeight

Sets the default height of the cells in the column.

Scrolling

Determines whether the column can be scrolled vertically if there are too many rows.

SEE ALSO

Activate, Alignment, ColumnWidth, DoubleClick, gadgets, Glyph, ReadOnly, RowHeight, Scrolling, Table, TableRow
TableRow

NAME

TableRow

PURPOSE

DSL gadget that defines a column in a Table gadget.

SYNOPSIS

Table {
    
    Contents {
        TableRow {
            Name = gadget-name;
            Title = title;
            
            ...  
        }
    }
}

DESCRIPTION

The TableRow gadget defines a row within a Table gadget. You must include one TableRow gadget for every fixed row in your table. For example, if your table always has ten rows, you must define ten TableRow gadgets.

This gadget can only be used as a subordinate of a Table gadget.

In addition to attributes common to all DSL gadgets, you can specify the following attributes for a TableRow gadget:

Activate Specifies a callback list that is invoked when the column is "activated" by the user.

Alignment Controls the alignment of the text in the column's cells.

ColumnWidth Specifies the default width of the cells in the row.

DoubleClick Defines a list of callbacks that are invoked when a user double-clicks on a cell in the row.

Glyph Associates a bitmap with the row, similar to BitmapTitle.
TableRow

ReadOnly Allows the user to browse the information in the row, but not alter it.
RowHeight Sets the height of the row.
Scrolling Determines whether the row can be scrolled horizontally if there are too many columns.

SEE ALSO
Activate, Alignment, ColumnWidth, DoubleClick, gadgets, Glyph, ReadOnly, RowHeight, Scrolling, Table, TableRow
TearOff

NAME

TearOff

PURPOSE

DSL attribute that controls whether a Menu gadget can be "torn off" of its parent dialog.

SYNOPSIS

TearOff = value

DESCRIPTION

The TearOff attribute controls whether a Menu gadget can be removed from its parent dialog and placed in another part of the display. When a user tears off a menu, it becomes its own moveable, resizeable window. The menu is still subordinate to the parent dialog; when the user closes the parent, the menu window closes too.

The attribute's value must be one of the following keywords:

YES The menu is a tear-off menu.
NO The menu is not a tear-off menu.

The default value is NO.

SEE ALSO

Menu
TellModify

NAME

TellModify

PURPOSE

DSL attribute that controls how unchanged values are passed to callback methods.

SYNOPSIS

TellModify = value

DESCRIPTION

The TellModify attribute dictates the how the dialog reports values for gadgets that were not changed.

The TellModify attribute is valid for the following gadgets: Choice, Collection, List, Page, Switch, Table, and Text.

Value

The TellModify attribute’s value must be one of the following keywords:

YES Passes the current value of the gadget, regardless of whether the user changed the value.

NO Passes the special value "__unchanged__" if the user left the value of the gadget unchanged.

The default value of the TellModify attribute is NO.

EXAMPLES

Text {
    Name = new_name;
    Title = "Node Name:"
    TellModify = YES;
    Value = $NEW_NODE_NAME;
}

SEE ALSO

Choice, Collection, List, Page, Switch, Table, Text,
NAME

Text

PURPOSE

DSL gadget that displays a single line of text the user can edit.

SYNTAX

Text {
  Name = gadget-name;
  Title = title;
  Value = "default-string";

  ...
}

DESCRIPTION

A Text gadget displays a single line of text the user can edit. In addition to the attributes common to all gadgets, you can specify the following attributes for a Text gadget:

- **Border** Controls whether a border is drawn around the gadget.
- **Changed** Specifies callbacks to be invoked when the user indicates finished changing the gadget’s value. The way the user indicates changes are complete varies according to the local “look-and-feel” of the Desktop. Common indicators are for the user to press <Return>, <Enter>, or <Tab>.
- **Columns** Specifies the width of the gadget.
- **Encrypt** Specifies whether the gadget’s value is to be encrypted before it is passed to callback methods. Encrypted text appears on the Desktop as a number of hollow boxes equal to the number of characters in the **Value** attribute.
- **TellModify** Specifies how the gadget’s value is passed to callback methods if the user does not change the value.
**Value**

Defines the initial text string that appears in the gadget. This string cannot contain newline ('\n') characters.

**EXAMPLES**

```plaintext
Text {
    Name = passwd;
    Title = "Password: ";
    Encrypt = CRYPT;
    Value = $USER_PASSWD;
}
```

**SEE ALSO**

Border, Changed, Columns, Encrypt, gadgets, TellModify, Value
Title

NAME

Title

PURPOSE

DSL attribute that defines the title of a gadget.

SYNOPSIS

Title = string

DESCRIPTION

The Title attribute specifies the title of a dialog or gadget. The title is used by administrators to identify gadgets on the screen. For example, the title of a CommandButton gadget is the label that appears inside the button. Titles are different from gadget names, which are the means by which an application references a gadget.

The Title attribute is valid on all gadget types, except Separator.

The attribute’s value can be any printable string. Its default value is the name of the dialog or gadget (defined by the Name attribute).

The position of a some gadget titles is controlled by the gadget’s TitlePos attribute.

EXAMPLES

CommandButton {
    Name = apply;
    Title = "Add Printer";
    Commands = add_print_entry@(...), dtc_dismiss(...) ;
}

SEE ALSO

Name, TitlePos
NAME
TitlePos

PURPOSE
DSL attribute that specifies the position of the gadget title.

SYNOPSIS
TitlePos = value

DESCRIPTION
The TitlePos attribute controls where a gadget’s title is displayed. The attribute is available on the following gadgets: Button, Choice, Collection, CommandButton, Group, List, Menu, Message, Page, Switch, Table, and Text.

The TitlePos attribute’s value must be one of the following keywords:

LEFT Displays the title to the left of the gadget.

RIGHT Displays the title to the right of the gadget.

TOP Displays the title directly above the gadget.

BOTTOM Displays the title immediately below the gadget.

SEE ALSO
Button, Choice, Collection, CommandButton, Group, List, Menu, Message, Page, Switch, Table, Text
TopRow

NAME

TopRow

PURPOSE

DSL attribute that specifies which row in a Table gadget is initially displayed as the top row.

SYNOPSIS

TopRow = value

DESCRIPTION

The TopRow attribute dictates which row in a Table gadget is initially displayed as the top row. Any rows above the specified top row can be viewed by scrolling upward.

This attribute is only defined for Table gadgets.

SEE ALSO

Table
NAME

Type

PURPOSE

DSL attribute that specifies the type of a gadget for drag-and-drop operations.

SYNOPSIS

Type = string

DESCRIPTION

The Type attribute specifies the type of an Icon or Collection gadget. The gadget type dictates what kinds of drag-and-drop operations are valid on the gadget.

The value of the Type attribute is a type name for which drag-and-drop operations are defined. There is no default value for this attribute.

SEE ALSO

Collection, Icon
Value

NAME

Value

PURPOSE

DSL attribute that assigns an initial value to a gadget.

SYNTAX

Value = value

DESCRIPTION

The Value attribute contains the initial data value of a gadget. For example, if you have a Text gadget that prompts the user to change a login name, you would probably set the gadget’s Value attribute to the existing login name.

Choice

The Value attribute contains the choice value of the default choice. The default value for Choice gadgets is the empty string (no choices are initially selected).

Collection

The Value attribute contains a comma-separated list of any icons that are selected in the collection. The default value for Collection gadgets is the empty string (no icons are initially selected).

List

The Value attribute contains the list of comma-separated choice values for the default selections. The default value for List gadgets is the empty string (no choices are initially selected).

Message

The Value attribute contains a string that is the message text to be displayed. This string can contain multiple lines of text separated by newline (‘\n’) characters. The default value for Message gadgets is the empty string.

Page

The Value attribute contains the current text string. This string can contain multiple lines of text separated by newline (‘\n’) characters. The default value for Page gadgets is the empty string.
Switch  The Value attribute contains the value of the On attribute if the switch is initially on; otherwise, it contains the value of the Off attribute. The default value is the character 1 (the switch appears "on").

Table  The attribute contains the value of a table cell that is initially selected.

Text  The Value attribute contains the current text string, which cannot contain newline ('\n') characters. The default value for Text gadgets is the empty string.

Example

Message {
  Name = md;
  Title = "Policy Region: ";
  Value = $HDO_MDO;
}

SEE ALSO

Choice, Collection, List, Message, Page, Switch, Table, Text
**ViewBy**

**NAME**

ViewBy

**PURPOSE**

DSL attribute that specifies how resources are displayed in a Collection gadget.

**SYNOPSIS**

ViewBy = value

**DESCRIPTION**

The ViewBy attribute controls how resources (Icon gadgets) appear in a Collection gadget. Resources can be viewed by their names or by their icons.

The attribute’s value must be one of the following keywords:

**ICON**

Causes the Desktop to display resources using their full icon bitmaps, with the names shown beneath the icons.

**TITLE**

Causes the Desktop to display resources with a small version of the icon bitmap to the left of the title. The small bitmap acts like a miniature icon that allows the user to drag the icon. By convention, icon bitmaps are normally 48 bits by 48 bits. If ViewBy is set to TITLE, TME 10 shrinks the icon bitmaps to 16 bits by 16 bits.

The ViewBy attribute’s default value is ICON.

**SEE ALSO**

Collection
NAME

Visible

PURPOSE

DSL attribute that specifies whether the current gadget is visible within the dialog.

SYNOPSIS

Visible = value

DESCRIPTION

The Visible attribute controls whether the current attribute is displayed in the current dialog. The attribute’s value can be either YES or NO. If Visible is NO, the gadget is not displayed in the dialog. This is not the same as being insensitive. An insensitive gadget appears in the dialog, but cannot be changed by the user.

The Visible attributes is available for all DSL gadgets.

SEE ALSO

Sensitive
Wrap

NAME

Wrap

PURPOSE

DSL attribute that controls text wrapping in a Page gadget.

SYNOPSIS

Wrap = value

DESCRIPTION

The Wrap attribute controls whether text lines in a Page gadget are wrapped if they are wider than the gadget width.

The attribute’s value must be one of the following keywords:

YES Enables text wrapping.

NO Disable text wrapping. Lines that are too long are truncated.

SEE ALSO

Page
NAME
dsl_colors

PURPOSE
Introduction to DSL color support.

DESCRIPTION
DSL supports color specification for dialogs and gadgets. The color can be specified using two attributes: Background and Foreground. Background controls the color of the dialog or gadget, and Foreground controls the color of anything drawn in the dialog or gadget. The TME supports the RGB color model, so you can select a color in two ways: by RGB color value or by name.

The value specification always takes the form #rrggb. The RGB color model specifies three components for each color: red, green, and blue. The rr variable specifies the red component, the gg specifies the green component, and bb specifies the blue component. The hexadecimal values specify the component intensity: 00 represents off and ff represents full on. For example, the DSL command below sets the Foreground attribute to green.

Foreground = #00ff00;

The green component is turned full on and the red and blue components are turned off.

All shades of all colors can be produced by changing the intensity of one or more color components. White is produced by turning all three components full on, and black is produced by turning all three components off. Other colors are produced by specifying some value between 00 and ff for each component. Light blue, for example, is produced by the following hexadecimal specification:

Foreground = #ADD8E6;

Each color component in this example is set to an intensity such that when they are combined, they produce light blue. The blue component has the greatest intensity, and thus the resulting color is perceived as a shade of blue.
Mixing the color components to obtain a specific color shade using hexadecimal values can be complex. You should therefore use this technique only when the required color is not available using the second technique. See your X users guide for more details about color specification and the RGB color model.

The second, and easier way of specifying a color in DSL is by color name. For example, to set the foreground to light blue, you would specify the following color name.

```
Foreground = lightblue;
```

DSL supports the following color keywords:

<table>
<thead>
<tr>
<th>Color</th>
<th>Color</th>
<th>Color</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>aliceblue</td>
<td>antiquewhite</td>
<td>antiquewhite1</td>
<td>antiquewhite2</td>
</tr>
<tr>
<td>antiquewhite3</td>
<td>antiquewhite4</td>
<td>aquamarine</td>
<td>aquamarine1</td>
</tr>
<tr>
<td>aquamarine2</td>
<td>aquamarine3</td>
<td>aquamarine4</td>
<td>azure</td>
</tr>
<tr>
<td>azure1</td>
<td>azure2</td>
<td>azure3</td>
<td>azure4</td>
</tr>
<tr>
<td>beige</td>
<td>bisque</td>
<td>bisque1</td>
<td>bisque2</td>
</tr>
<tr>
<td>bisque3</td>
<td>bisque4</td>
<td>black</td>
<td>blanchedalmond</td>
</tr>
<tr>
<td>blue</td>
<td>blue1</td>
<td>blue2</td>
<td>blue3</td>
</tr>
<tr>
<td>blue4</td>
<td>blueviolet</td>
<td>brown</td>
<td>brown1</td>
</tr>
<tr>
<td>brown2</td>
<td>brown3</td>
<td>brown4</td>
<td>burlywood</td>
</tr>
<tr>
<td>burlywood1</td>
<td>burlywood2</td>
<td>burlywood3</td>
<td>burlywood4</td>
</tr>
<tr>
<td>cadetblue</td>
<td>cadetblue1</td>
<td>cadetblue2</td>
<td>cadetblue3</td>
</tr>
<tr>
<td>cadetblue4</td>
<td>chartreuse</td>
<td>chartreuse1</td>
<td>chartreuse2</td>
</tr>
<tr>
<td>chartreuse3</td>
<td>chartreuse4</td>
<td>chocolate</td>
<td>chocolate1</td>
</tr>
<tr>
<td>chocolate2</td>
<td>chocolate3</td>
<td>chocolate4</td>
<td>coral</td>
</tr>
<tr>
<td>coral1</td>
<td>coral2</td>
<td>coral3</td>
<td>coral4</td>
</tr>
<tr>
<td>cornflowerblue</td>
<td>cornsilk</td>
<td>cornsilk1</td>
<td>cornsilk2</td>
</tr>
<tr>
<td>cornsilk3</td>
<td>cornsilk4</td>
<td>cyan</td>
<td>cyan1</td>
</tr>
<tr>
<td>cyan2</td>
<td>cyan3</td>
<td>cyan4</td>
<td>darkgoldenrod</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>darkgoldenrod1</td>
<td>darkgoldenrod2</td>
<td>darkgoldenrod3</td>
<td>darkgoldenrod4</td>
</tr>
<tr>
<td>darkgreen</td>
<td>darkkhaki</td>
<td>darkolivegreen</td>
<td>darkolivegreen1</td>
</tr>
<tr>
<td>darkolivegreen2</td>
<td>darkolivegreen3</td>
<td>darkolivegreen4</td>
<td>darkorange</td>
</tr>
<tr>
<td>darkorange1</td>
<td>darkorange2</td>
<td>darkorange3</td>
<td>darkorange4</td>
</tr>
<tr>
<td>darkorchid</td>
<td>darkorchid1</td>
<td>darkorchid2</td>
<td>darkorchid3</td>
</tr>
<tr>
<td>darkorchid4</td>
<td>darksalmon</td>
<td>darkseagreen</td>
<td>darkseagreen1</td>
</tr>
<tr>
<td>darkseagreen2</td>
<td>darkseagreen3</td>
<td>darkseagreen4</td>
<td>darkslateblue</td>
</tr>
<tr>
<td>darkslategray</td>
<td>darkslategray1</td>
<td>darkslategray2</td>
<td>darkslategray3</td>
</tr>
<tr>
<td>darkslategray4</td>
<td>darkslategrey</td>
<td>darkturquoise</td>
<td>darkviolet</td>
</tr>
<tr>
<td>deepink</td>
<td>deepink1</td>
<td>deepink2</td>
<td>deepink3</td>
</tr>
<tr>
<td>deepink4</td>
<td>deepskyblue</td>
<td>deepskyblue1</td>
<td>deepskyblue2</td>
</tr>
<tr>
<td>deepskyblue3</td>
<td>deepskyblue4</td>
<td>dimgray</td>
<td>dimgrey</td>
</tr>
<tr>
<td>dodgerblue</td>
<td>dodgerblue1</td>
<td>dodgerblue2</td>
<td>dodgerblue3</td>
</tr>
<tr>
<td>dodgerblue4</td>
<td>firebrick</td>
<td>firebrick1</td>
<td>firebrick2</td>
</tr>
<tr>
<td>firebrick3</td>
<td>firebrick4</td>
<td>floralwhite</td>
<td>forestgreen</td>
</tr>
<tr>
<td>gainsboro</td>
<td>ghostwhite</td>
<td>gold</td>
<td>gold1</td>
</tr>
<tr>
<td>gold2</td>
<td>gold3</td>
<td>gold4</td>
<td>goldenrod</td>
</tr>
<tr>
<td>goldenrod1</td>
<td>goldenrod2</td>
<td>goldenrod3</td>
<td>goldenrod4</td>
</tr>
<tr>
<td>gray</td>
<td>gray0</td>
<td>gray1</td>
<td>gray10</td>
</tr>
<tr>
<td>gray100</td>
<td>gray11</td>
<td>gray12</td>
<td>gray13</td>
</tr>
<tr>
<td>gray14</td>
<td>gray15</td>
<td>gray16</td>
<td>gray17</td>
</tr>
<tr>
<td>gray18</td>
<td>gray19</td>
<td>gray2</td>
<td>gray20</td>
</tr>
<tr>
<td>gray21</td>
<td>gray22</td>
<td>gray23</td>
<td>gray24</td>
</tr>
<tr>
<td>gray25</td>
<td>gray26</td>
<td>gray27</td>
<td>gray28</td>
</tr>
<tr>
<td>dsl_colors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray29</td>
<td>gray3</td>
<td>gray30</td>
<td>gray31</td>
</tr>
<tr>
<td>gray32</td>
<td>gray33</td>
<td>gray34</td>
<td>gray35</td>
</tr>
<tr>
<td>gray36</td>
<td>gray37</td>
<td>gray38</td>
<td>gray39</td>
</tr>
<tr>
<td>gray4</td>
<td>gray40</td>
<td>gray41</td>
<td>gray42</td>
</tr>
<tr>
<td>gray43</td>
<td>gray44</td>
<td>gray45</td>
<td>gray46</td>
</tr>
<tr>
<td>gray47</td>
<td>gray48</td>
<td>gray49</td>
<td>gray5</td>
</tr>
<tr>
<td>gray50</td>
<td>gray51</td>
<td>gray52</td>
<td>gray53</td>
</tr>
<tr>
<td>gray54</td>
<td>gray55</td>
<td>gray56</td>
<td>gray57</td>
</tr>
<tr>
<td>gray58</td>
<td>gray59</td>
<td>gray6</td>
<td>gray60</td>
</tr>
<tr>
<td>gray61</td>
<td>gray62</td>
<td>gray63</td>
<td>gray64</td>
</tr>
<tr>
<td>gray65</td>
<td>gray66</td>
<td>gray67</td>
<td>gray68</td>
</tr>
<tr>
<td>gray69</td>
<td>gray7</td>
<td>gray70</td>
<td>gray71</td>
</tr>
<tr>
<td>gray72</td>
<td>gray73</td>
<td>gray74</td>
<td>gray75</td>
</tr>
<tr>
<td>gray76</td>
<td>gray77</td>
<td>gray78</td>
<td>gray79</td>
</tr>
<tr>
<td>gray8</td>
<td>gray80</td>
<td>gray81</td>
<td>gray82</td>
</tr>
<tr>
<td>gray83</td>
<td>gray84</td>
<td>gray85</td>
<td>gray86</td>
</tr>
<tr>
<td>gray87</td>
<td>gray88</td>
<td>gray89</td>
<td>gray9</td>
</tr>
<tr>
<td>gray90</td>
<td>gray91</td>
<td>gray92</td>
<td>gray93</td>
</tr>
<tr>
<td>gray94</td>
<td>gray95</td>
<td>gray96</td>
<td>gray97</td>
</tr>
<tr>
<td>gray98</td>
<td>gray99</td>
<td>green</td>
<td>green1</td>
</tr>
<tr>
<td>green2</td>
<td>green3</td>
<td>green4</td>
<td>greenyellow</td>
</tr>
<tr>
<td>grey</td>
<td>grey0</td>
<td>grey1</td>
<td>grey10</td>
</tr>
<tr>
<td>grey100</td>
<td>grey11</td>
<td>grey12</td>
<td>grey13</td>
</tr>
<tr>
<td>grey14</td>
<td>grey15</td>
<td>grey16</td>
<td>grey17</td>
</tr>
<tr>
<td>grey18</td>
<td>grey19</td>
<td>grey2</td>
<td>grey20</td>
</tr>
<tr>
<td>grey21</td>
<td>grey22</td>
<td>grey23</td>
<td>grey24</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>grey25</td>
<td>grey26</td>
<td>grey27</td>
<td>grey28</td>
</tr>
<tr>
<td>grey29</td>
<td>grey3</td>
<td>grey30</td>
<td>grey31</td>
</tr>
<tr>
<td>grey32</td>
<td>grey33</td>
<td>grey34</td>
<td>grey35</td>
</tr>
<tr>
<td>grey36</td>
<td>grey37</td>
<td>grey38</td>
<td>grey39</td>
</tr>
<tr>
<td>grey4</td>
<td>grey40</td>
<td>grey41</td>
<td>grey42</td>
</tr>
<tr>
<td>grey43</td>
<td>grey44</td>
<td>grey45</td>
<td>grey46</td>
</tr>
<tr>
<td>grey47</td>
<td>grey48</td>
<td>grey49</td>
<td>grey5</td>
</tr>
<tr>
<td>grey50</td>
<td>grey51</td>
<td>grey52</td>
<td>grey53</td>
</tr>
<tr>
<td>grey54</td>
<td>grey55</td>
<td>grey56</td>
<td>grey57</td>
</tr>
<tr>
<td>grey58</td>
<td>grey59</td>
<td>grey6</td>
<td>grey60</td>
</tr>
<tr>
<td>grey61</td>
<td>grey62</td>
<td>grey63</td>
<td>grey64</td>
</tr>
<tr>
<td>grey65</td>
<td>grey66</td>
<td>grey67</td>
<td>grey68</td>
</tr>
<tr>
<td>grey69</td>
<td>grey7</td>
<td>grey70</td>
<td>grey71</td>
</tr>
<tr>
<td>grey72</td>
<td>grey73</td>
<td>grey74</td>
<td>grey75</td>
</tr>
<tr>
<td>grey76</td>
<td>grey77</td>
<td>grey78</td>
<td>grey79</td>
</tr>
<tr>
<td>grey8</td>
<td>grey80</td>
<td>grey81</td>
<td>grey82</td>
</tr>
<tr>
<td>grey83</td>
<td>grey84</td>
<td>grey85</td>
<td>grey86</td>
</tr>
<tr>
<td>grey87</td>
<td>grey88</td>
<td>grey89</td>
<td>grey9</td>
</tr>
<tr>
<td>grey90</td>
<td>grey91</td>
<td>grey92</td>
<td>grey93</td>
</tr>
<tr>
<td>grey94</td>
<td>grey95</td>
<td>grey96</td>
<td>grey97</td>
</tr>
<tr>
<td>grey98</td>
<td>grey99</td>
<td>honeydew</td>
<td>honeydew1</td>
</tr>
<tr>
<td>honeydew2</td>
<td>honeydew3</td>
<td>honeydew4</td>
<td>hotpink</td>
</tr>
<tr>
<td>hotpink1</td>
<td>hotpink2</td>
<td>hotpink3</td>
<td>hotpink4</td>
</tr>
<tr>
<td>indianred</td>
<td>indianred1</td>
<td>indianred2</td>
<td>indianred3</td>
</tr>
</tbody>
</table>
### dsl_colors

<table>
<thead>
<tr>
<th>Color</th>
<th>Color</th>
<th>Color</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>indianred4</td>
<td>ivory</td>
<td>ivory1</td>
<td>ivory2</td>
</tr>
<tr>
<td>ivory3</td>
<td>ivory4</td>
<td>khaki</td>
<td>khaki1</td>
</tr>
<tr>
<td>khaki2</td>
<td>khaki3</td>
<td>khaki4</td>
<td>lavender</td>
</tr>
<tr>
<td>lavenderblush</td>
<td>lavenderblush1</td>
<td>lavenderblush2</td>
<td>lavenderblush3</td>
</tr>
<tr>
<td>lavenderblush4</td>
<td>lawngreen</td>
<td>lemonchiffon</td>
<td>lemonchiffon1</td>
</tr>
<tr>
<td>lemonchiffon2</td>
<td>lemonchiffon3</td>
<td>lemonchiffon4</td>
<td>lightblue</td>
</tr>
<tr>
<td>lightblue1</td>
<td>lightblue2</td>
<td>lightblue3</td>
<td>lightblue4</td>
</tr>
<tr>
<td>lightcoral</td>
<td>lightcyan</td>
<td>lightcyan1</td>
<td>lightcyan2</td>
</tr>
<tr>
<td>lightcyan3</td>
<td>lightcyan4</td>
<td>lightgoldenrod</td>
<td>lightgoldenrod1</td>
</tr>
<tr>
<td>lightgoldenrod2</td>
<td>lightgoldenrod3</td>
<td>lightgoldenrod4</td>
<td>lightgoldenrodyellow</td>
</tr>
<tr>
<td>lightgray</td>
<td>lightgrey</td>
<td>lightpink</td>
<td>lightpink1</td>
</tr>
<tr>
<td>lightpink2</td>
<td>lightpink3</td>
<td>lightpink4</td>
<td>lightsalmon</td>
</tr>
<tr>
<td>lightsalmon1</td>
<td>lightsalmon2</td>
<td>lightsalmon3</td>
<td>lightsalmon4</td>
</tr>
<tr>
<td>lightseagreen</td>
<td>lightskyblue</td>
<td>lightskyblue1</td>
<td>lightskyblue2</td>
</tr>
<tr>
<td>lightskyblue3</td>
<td>lightskyblue4</td>
<td>lightslateblue</td>
<td>lightslategray</td>
</tr>
<tr>
<td>lightslategrey</td>
<td>lightsteelblue</td>
<td>lightsteelblue1</td>
<td>lightsteelblue2</td>
</tr>
<tr>
<td>lightsteelblue3</td>
<td>lightsteelblue4</td>
<td>lightyellow</td>
<td>lightyellow1</td>
</tr>
<tr>
<td>lightyellow2</td>
<td>lightyellow3</td>
<td>lightyellow4</td>
<td>limegreen</td>
</tr>
<tr>
<td>linen</td>
<td>magenta</td>
<td>magenta1</td>
<td>magenta2</td>
</tr>
<tr>
<td>magenta3</td>
<td>magenta4</td>
<td>maroon</td>
<td>maroon1</td>
</tr>
<tr>
<td>maroon2</td>
<td>maroon3</td>
<td>maroon4</td>
<td>mediumaquamarine</td>
</tr>
<tr>
<td>mediumblue</td>
<td>mediumorchid</td>
<td>mediumorchid1</td>
<td>mediumorchid2</td>
</tr>
<tr>
<td>mediumorchid3</td>
<td>mediumorchid4</td>
<td>mediumpurple</td>
<td>mediumpurple1</td>
</tr>
<tr>
<td>mediumpurple2</td>
<td>mediumpurple3</td>
<td>mediumpurple4</td>
<td>mediumseagreen</td>
</tr>
<tr>
<td>mediumslateblue</td>
<td>mediumspringgreen</td>
<td>mediumturquoise</td>
<td>mediumvioletred</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>Color</td>
<td>Color</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>midnightblue</td>
<td>mintcream</td>
<td>mistyrose</td>
<td>mistyrose</td>
</tr>
<tr>
<td>mistyrose2</td>
<td>mistyrose3</td>
<td>mistyrose4</td>
<td>moccasin</td>
</tr>
<tr>
<td>navajowhite</td>
<td>navajowhite1</td>
<td>navajowhite2</td>
<td>navajowhite3</td>
</tr>
<tr>
<td>navajowhite4</td>
<td>navy</td>
<td>navyblue</td>
<td>oldlace</td>
</tr>
<tr>
<td>olivedrab</td>
<td>olivedrab1</td>
<td>olivedrab2</td>
<td>olivedrab3</td>
</tr>
<tr>
<td>olivedrab4</td>
<td>orange</td>
<td>orange1</td>
<td>orange2</td>
</tr>
<tr>
<td>orange3</td>
<td>orange4</td>
<td>orangered</td>
<td>orangered1</td>
</tr>
<tr>
<td>orangered2</td>
<td>orangered3</td>
<td>orangered4</td>
<td>orchid</td>
</tr>
<tr>
<td>orchid1</td>
<td>orchid2</td>
<td>orchid3</td>
<td>orchid4</td>
</tr>
<tr>
<td>palegoldenrod</td>
<td>palegreen</td>
<td>palegreen1</td>
<td>palegreen2</td>
</tr>
<tr>
<td>palegreen3</td>
<td>palegreen4</td>
<td>paleturquoise</td>
<td>paleturquoise1</td>
</tr>
<tr>
<td>paleturquoise2</td>
<td>paleturquoise3</td>
<td>paleturquoise4</td>
<td>palevioletred</td>
</tr>
<tr>
<td>palevioletred1</td>
<td>palevioletred2</td>
<td>palevioletred3</td>
<td>palevioletred4</td>
</tr>
<tr>
<td>papayawhip</td>
<td>peachpuff</td>
<td>peachpuff1</td>
<td>peachpuff2</td>
</tr>
<tr>
<td>peachpuff3</td>
<td>peachpuff4</td>
<td>peru</td>
<td>pink</td>
</tr>
<tr>
<td>pink1</td>
<td>pink2</td>
<td>pink3</td>
<td>pink4</td>
</tr>
<tr>
<td>plum</td>
<td>plum1</td>
<td>plum2</td>
<td>plum3</td>
</tr>
<tr>
<td>plum4</td>
<td>powderblue</td>
<td>purple</td>
<td>purple1</td>
</tr>
<tr>
<td>purple2</td>
<td>purple3</td>
<td>purple4</td>
<td>red</td>
</tr>
<tr>
<td>red1</td>
<td>red2</td>
<td>red3</td>
<td>red4</td>
</tr>
<tr>
<td>rosybrown</td>
<td>rosybrown1</td>
<td>rosybrown2</td>
<td>rosybrown3</td>
</tr>
<tr>
<td>rosybrown4</td>
<td>royalblue</td>
<td>royalblue1</td>
<td>royalblue2</td>
</tr>
<tr>
<td>royalblue3</td>
<td>royalblue4</td>
<td>saddlebrown</td>
<td>salmon</td>
</tr>
<tr>
<td>salmon1</td>
<td>salmon2</td>
<td>salmon3</td>
<td>salmon4</td>
</tr>
<tr>
<td>sandybrown</td>
<td>seagreen</td>
<td>seagreen1</td>
<td>seagreen2</td>
</tr>
</tbody>
</table>
### dsl_colors

<table>
<thead>
<tr>
<th>seagreen3</th>
<th>seagreen4</th>
<th>seashell</th>
<th>seashell1</th>
</tr>
</thead>
<tbody>
<tr>
<td>seashell2</td>
<td>seashell3</td>
<td>seashell4</td>
<td>sienna</td>
</tr>
<tr>
<td>sienna1</td>
<td>sienna2</td>
<td>sienna3</td>
<td>sienna4</td>
</tr>
<tr>
<td>skyblue</td>
<td>skyblue1</td>
<td>skyblue2</td>
<td>skyblue3</td>
</tr>
<tr>
<td>skyblue4</td>
<td>slateblue</td>
<td>slateblue1</td>
<td>slateblue2</td>
</tr>
<tr>
<td>slateblue3</td>
<td>slateblue4</td>
<td>slategray</td>
<td>slategray1</td>
</tr>
<tr>
<td>slategray2</td>
<td>slategray3</td>
<td>slategray4</td>
<td>slategrey</td>
</tr>
<tr>
<td>snow</td>
<td>snow1</td>
<td>snow2</td>
<td>snow3</td>
</tr>
<tr>
<td>snow4</td>
<td>springgreen</td>
<td>springgreen1</td>
<td>springgreen2</td>
</tr>
<tr>
<td>springgreen3</td>
<td>springgreen4</td>
<td>steelblue</td>
<td>steelblue1</td>
</tr>
<tr>
<td>steelblue2</td>
<td>steelblue3</td>
<td>steelblue4</td>
<td>tan</td>
</tr>
<tr>
<td>tan1</td>
<td>tan2</td>
<td>tan3</td>
<td>tan4</td>
</tr>
<tr>
<td>thistle</td>
<td>thistle1</td>
<td>thistle2</td>
<td>thistle3</td>
</tr>
<tr>
<td>thistle4</td>
<td>tomato</td>
<td>tomato1</td>
<td>tomato2</td>
</tr>
<tr>
<td>tomato3</td>
<td>tomato4</td>
<td>turquoise</td>
<td>turquoise1</td>
</tr>
<tr>
<td>turquoise2</td>
<td>turquoise3</td>
<td>turquoise4</td>
<td>violet</td>
</tr>
<tr>
<td>violetred</td>
<td>violetred1</td>
<td>violetred2</td>
<td>violetred3</td>
</tr>
<tr>
<td>violetred4</td>
<td>wheat</td>
<td>wheat1</td>
<td>wheat2</td>
</tr>
<tr>
<td>wheat3</td>
<td>wheat4</td>
<td>white</td>
<td>whitesmoke</td>
</tr>
<tr>
<td>yellow</td>
<td>yellow1</td>
<td>yellow2</td>
<td>yellow3</td>
</tr>
</tbody>
</table>

**SEE ALSO**

Background, Foreground
NAME
gadgets

PURPOSE
Introduction to DSL gadgets.

DESCRIPTION
Dialog gadgets define the contents of dialogs. There are three categories of gadget: command, value, and inactive.

Command gadgets invoke operations when they are selected. The following DSL gadgets are command gadgets.

**Button**
Invokes an operation when the button is selected.

**CommandButton**
Button that is presented at the bottom of a dialog.

**Icon**
Presents a visual representation of an object.

Users can operate on the object by manipulating the gadget.

Value gadgets display one or more values and allow the user to change the gadget’s value. The following are DSL value gadgets:

**Choice**
Presents a list of choices and allows a maximum of one entry to be selected.

**List**
Presents a list of choices and allows multiple entries to be selected.

**Page**
Presents a multi-line text field that the user can edit.

**Switch**
Presents a simple two-state switch.

**Table**
Presents a multi-row, multi-column editable table.

**Text**
Presents a one-line text string that the user can edit.

Inactive gadgets can only display data or contain other gadgets. The following are inactive gadgets:

**Collection**
Presents a dialog sector containing a collection of member icon gadgets.
gadgets

**Group**

Presents a set of gadgets as a group. A group gadget can contain most other gadgets, including other groups.

**Menu**

Presents a menu of **Button** gadgets.

**Message**

Presents a non-editable text string.

**Separator**

Presents a single horizontal or vertical separating line.

Every gadget has as set of attributes that allow to customize the gadget’s characteristics. When defining a attributes for a gadget, you can simply declare the attribute inside the gadget specification.

Attribute keywords are not reserved for a particular dialog or gadget type; they frequently specify similar properties for dialogs and several types of gadgets. The **Name** attribute, for example, can be used to specify the name of a command dialog or any gadget.

The following attributes are available to all DSL gadgets, except **Separator**. The **Separator** gadget supports a subset of these attributes.

**Background**

Specifies the background color of a gadget.

**BitmapTitle**

Specifies a bitmap to be used as a gadget’s title.

**ChildColumnAlignment**

Specifies the horizontal alignment of a gadget that is in a dialog or group that uses grid alignment.

**ChildRowAlignment**

Specifies the vertical alignment of a gadget that is in a dialog or group that uses grid alignment.

**Font**

Specifies the default font for a gadget.

**Foreground**

Specifies the foreground color of a gadget.

**GridHorizontal**

Specifies the relative horizontal location of a gadget that is in a dialog or group that uses grid alignment.

**GridVertical**

Specifies the relative vertical location of a gadget that is in a dialog or group that uses grid alignment.

**HelpMessage**

Specifies a help message that is displayed when a user points at a gadget.
Name  Specifies the internal name of a gadget.
Sensitive  Specifies whether a gadget is sensitive to user input.
Title  Specifies the title of a gadget.
Visible  Specifies whether a gadget is visible on a dialog.

Most gadgets support a superset of the attributes listed above. Consult each gadget's manual page for a list of additional supported attributes.
Index

A
Accelerator gadget attribute  4–9,  C-2
actions  
  adding custom  10–5,  B–5
  defined
  getting the definition of  10–6
  listing  10–7,  B–20,  B–29,  B–30,  B–32
  listing pending  10–8
  passing properties to  10–6
  pre- and post-distribution  10–5
  removing custom  10–9,  B–52
Activate gadget attribute  4–9,  C-3
Alignment gadget attribute  4–9,  C-4
AppendText gadget attribute  4–9,  C-6
attributes  
  adding custom  6–1,  B–7
  customization example  11-1
  defined  2–1
  getting the value of  6–2,  B–22
  lazy propagation  6–2
  relationship to dialogs  6–1
  removing custom  6–3,  B–54
  setting the value of  6–3,  B–68,  B–69
B  
Background gadget attribute  4–9,  4–32,  
  C-7
bitmap format. See XPM bitmap format
Bitmap gadget attribute  C-8
BITMAP_FILE directive  4–32
bitmaps  
  converting to XPM  9–2,  B–11
  defined  4–31
  format  4–32
  installing custom  9–8,  B–40
  installing in remote TMRs  9–9
  installing on an instance  9–10
  installing on create dialogs  9–10
  listing  9–14,  B–34
  removing custom  9–11,  9–13,  B–55
  removing from remote TMRs  9–12,  
    9–14
  retrieving  9–6,  B–23
  uses  9–1
BitmapTitle gadget attribute  4–9,  4–32,  
  9–4,  C-9
blocks  
  Attribute  4–8
  Gadgets  4–14
  Variable  4–7
Border gadget attribute  4–10,  C-10
Bourne shell  7–1,  10–5
browser  C-84
Button gadget  4–15,  C-11

C
C locale  
  described  4–40
callbacks  
  ACLs  7–8
  arguments  7–2
  call syntax  7–1
  confirmation dialogs in  7–17
  defined  3–2,  4–21
  error dialogs in  7–19
  error handling  4–31
  example customization  11-6,  11-16
  exit codes  7–2
  gadget library
    example  4–25
identification of 4–25
introduction 4–25
installing custom 7–4, 7–10, B–48
installing in remote TMRs 7–9
introduction 3–9
message dialogs in 7–19
method paths 7–4
platform-specific 7–6, 7–14
purpose 7–1
reading standard input 7–2
removing custom 7–14, B–61
results 4–7
specification of
gadget library 4–25–??
support utilities 7–17
catalog sets 4–41
CellBorderStyle gadget attribute C-12
Changed gadget attribute 4–10, C-13
ChildColumnAlignment gadget attribute
4–10, 4–61, 4–64, C-14
ChildRowAlignment gadget attribute
4–10, 4–61, 4–64, C-16
Choice gadget 4–16, 4–50, C-18
choice lists. See lists
Choices gadget attribute 4–10, C-20
CLI commands
getdialog 5–3
rdsl 4–47, 5–6, 9–20
waddaction 10–5, 10–9, B–5
waddattr 6–1, 11–1, B–7
wadpolm 8–2, 8–3, B–8
wadprop 10–2, B–10
wconbitmap 9–2, 9–3, B–11
wdispconf 7–17, B–13
wdisperr 7–19, B–14
wdispmsg 7–19, B–15, B–16, B–18
wgetaction 10–6, B–20
wgetattr 6–2, B–22
wgetbitmap 9–6, 9–7, 9–19, B–23
wgetdialog 5–4, 5–5, 9–19, B–26
wgetpending 10–8, B–29
wlsactions 10–7, B–30, B–32
wlsbitmap 9–14, B–34
wlsdialog 5–12, B–36
wlsicon 9–29, B–38
wputbitmap 9–8, 9–10, 9–11, 9–22, B–40
wputdialog 5–7, 5–8, 5–9, 9–22, B–43
wputicon 9–21, 9–28, B–46
wputmeth 7–4, 7–9, 7–11, 7–14, B–48
wputstate 9–23, B–51
wrmaction 10–9, B–52
wrmattr 6–3, B–54
wrmbitmap 9–11, 9–12, 9–13, B–55
wrmdialog 5–9, 5–10, 5–11, B–57
wrmicon 9–28, B–59
wrmmeth 7–14, 7–15, 7–16, B–61
wrmpolm 8–3, B–63
wrmprop 10–4, B–65
wrumpolm 8–4, B–66
wsattr 6–3, B–68, B–69
wstatusline 7–25, B–70
CMsg DSL type 4–7
Collection gadget 4–16, C-21
color 4–32
Columns gadget attribute 4–10, C-25
ColumnSelection gadget attribute 4–10, C-23
ColumnWidth gadget attribute 4–10, C-24
Command dialogs. See dialogs
CommandButton gadget C-26
Commands gadget attribute 4–10, C-30
confirmation dialogs 7–17, B–13
CopyIn gadget attribute 4–10, C-31
create dialogs
bitmaps  9–7, 9–10, 9–13
custom callbacks  7–11, 7–16
custom dialogs  5–5, 5–9, 5–11
CString DSL type  4–7
customization
typical composition  2–5

D
Default dialog attribute  4–10, C-33
descriptors. See dialog descriptors
desktop  3–1
desktop commands  3–9, 4–24
dialog callbacks. See callbacks
dialog descriptors
customizing instances  5–8
defined  5–1
installing custom  5–7, 5–9
installing in remote TMRs  5–8
listing  5–12
removing custom  5–9, 5–11
removing from remote TMRs  5–10
retrieving  5–3, 5–5
reverse compiling  5–6, B–4
dialogs
attributes  3–6, 4–8
bitmaps in  4–31
blocks  4–6
Command  4–6, C-27
create  5–5
defining a help menu  C-48
defining help text  C-49
dialog variables  4–7
example specification  4–49
installing custom  5–7, 5–8, 5–9,
B–43
installing in remote TMRs  5–8
listing  5–12, B–36
name  C-65
previewing  4–46
purpose  C-73
removing custom  5–9, 5–11, B–57
removing from remote TMRs  5–10
resource-wide customization  5–4
retrieving dialog descriptor  5–3, 5–5,
B–26
reverse compiling  4–47, 5–6, B–4
setting default action  C-33
setting default font  C-38
states  3–5
status field C-87, C-88
style  C-34
system variables  4–8
title  C-102
title bitmap  C-9
variables  3–6, 4–46
DialogStyle dialog attribute  C-34
DialogStyle gadget attribute  4–10
DoubleClick gadget attribute  4–11, C-35
drag-and-drop rules  C-31, C-61, C-63,
C-105
DSL
Attribute block  4–8
blocks  4–5
Button gadget  C-11
Choice gadget  C-18
Collection gadget  C-21
color support  4–32, C-7, C-39
Command dialogs  C-27
CommandButton gadget  C-26
declaring gadgets  4–14
dialog attributes  4–8
dsl command  B–2
gadget attribute  4–8
gadgets  C-119
Gadgets block  4–14
Group gadget  C-46
Icon gadget  C-50
List gadget  C-55
logical organization  4–5
Menu gadget  C-57
Message gadget C-59
msg directive
  example 4–43
  syntax 4–43
  syntax variables 4–43
Page gadget C-70
rdsl command B–4
reverse compiling descriptors 4–47
Separator gadget C-82
strings 4–21
Switch gadget C-89
Table gadget C-91
TableColumn gadget C-94
TableRow gadget C-96
Text gadget C-100
variable declaration 4–7
Variables block 4–7
dsl command B–2
DSL preprocessor 4–46

Encrypt gadget attribute 4–11, C-36
encryption 4–11, C-36
error dialogs 7–19, B–14

Font gadget attribute 4–11, C-38
Foreground gadget attribute 4–11, 4–32, C-39
foreign languages
developing applications in 4–40

Gadget library callbacks
  example 4–25
indentification of 4–25
introduction 4–25
gadgets
  active 4–15
  alignment 4–51, C-4
  attributes 4–8
  borders on C-10
  button 4–15
  declaration 4–14
  defined 3–6
  defining help C-49
  grid alignment 4–61
  groups 4–51, C-46
  icon 4–15
  inactive 4–16
  name C-65
  sensitivity C-81
  setting background color C-7
  setting default font C-38
  setting foreground color C-39
  setting gadget height C-78
  setting width C-25
  tables 4–16
  title C-102
  title bitmap C-9
  title position C-103
  value C-106
Glyph gadget attribute 4–11, C-41
grid alignment 4–61, 4–62
  horizontal alignment C-14
  horizontal position C-43
Layout gadget attribute C-52
  setting grid height C-42
  setting grid width C-45
  vertical alignment C-16
  vertical position C-44
GridHeight gadget attribute 4–61, 4–64, C-42
GridHorizontal gadget alignment C-43
GridHorizontal gadget attribute 4–11, 4–61, 4–64
GridVertical gadget attribute 4–11, 4–61, 4–64, C-44
GridWidth gadget attribute 4–61, 4–64, C-45
Group gadget 4–16, C-46

H
Help gadget attribute 4–11
HelpMenu dialog attribute C-48
HelpMenu gadget attribute 4–11
HelpMessage gadget attribute 4–11, C-49

I
Icon gadget 4–15, C-50
Iconic gadget attribute 4–11
icons 4–15
  associated bitmap 9–17, 9–19, C-8
  associated dialog 9–19
  associated object C-66
  changing resource states 9–23, B–51
  customizing 9–20
  defined 9–1
  defining new states 9–21, B–46
  dialog 9–17
  example menu customization 11–10
  getting current state 9–24
Icon gadget C-50
listing 9–29, B–38
removing custom 9–28, B–59
replacing bitmap of 9–22
replacing dialog of 9–22
type name C-105
inactive gadgets 4–16
instances 2–3

L
language environment
described 4–40
Layout gadget attribute 4–11, 4–62, C-14, C-16, C-52
LayoutPolicy gadget attribute 4–11, C-53
lazy propagation 6–2, 11–1
LeftColumn gadget attribute 4–12, C-54
legacy callbacks
  see callbacks
List gadget 4–16, C-55
lists
  browser C-84
  Choice gadget C-18
  example of 4–50
  List gadget C-55
  populating C-20
  sorting elements C-86
  specifying format C-83
locales
described 4–40

M
main dialog block 4–6
Menu gadget 4–16, C-57
menus
  accelerators C-2, C-60
  choices C-11
  Menu gadget C-57
  pull-down 5–13
  tear-off C-98
message catalogs
described 4–40
directives 4–41
DSL 4–41
sets 4–41
variables 4–41
message dialogs 7–19, B–15, B–16, B–18
Message gadget 4–16, C-59
message variables 4–41
methods
  callback 3–2
  policy 8–1
Mnemonic gadget attribute 4–12, C-60
MoveIn gadget attribute 4–12, C-61
MoveOut gadget attribute 4–12, C-63
msg_cat directory 4–42

N
Name gadget attribute 4–6, 4–12, 4–49, C-65

O
Object gadget attribute 4–12, C-66
objects 2–1
Off gadget attribute 4–12, C-67
On gadget attribute 4–12, C-68
Open gadget attribute 4–12, C-69

P
Page gadget 4–16, C-70
PBMPLUS bitmap filters 9–2
per-instance customizations
  bitmaps 9–7, 9–10, 9–13
callbacks 7–10
dialogs 5–5, 5–8, 5–11
removing callbacks 7–15
pixmap format. See XPM bitmap format
policy
  customizing 8–1
  in profile-based applications 8–1
policy methods 8–1
adding custom B–8
default 8–1
installing custom 8–2
intrinsic and script 8–2
invoking 8–4, B–66
naming conventions 8–2
removing custom 8–3, B–63
validation 8–1
PopDown dialog attribute C-72
PopDown gadget attribute 4–12
profile actions
  see actions
profile properties
  see properties
profiles 8–1
  customizing profile dialogs 10–9
  introduction 10–1
properties
  adding custom 10–2, B–10
defined 10–2
dialog support for 10–9
removing B–65
removing custom 10–4
pull-down menus 5–13, 11–12
Purpose dialog attribute C-73
Purpose gadget attribute 4–12
rdsl command 4–47, 5–6, 9–20, B–4
ReadOnly gadget attribute 4–13, C-74
Refresh dialog attribute 4–13, C-74
resource states
  changing 9–23, B–51
defined 9–15
defining B–46
getting 9–24
listing 9–29
resource types
  attributes 2–1
behavior component 2–3
custom bitmaps 9–9, 9–12
custom callbacks 7–4
defined 2–1
extension component 2–3, 5–2, 9–5
instances 2–3
listing dialogs for 5–12
removing custom callbacks 7–14
retrieving bitmaps 9–6
user interface component 2–3, 5–2, 9–5
RGB color model 4–33
RowHeight gadget attribute C-76
Rows gadget attribute 4–13, C-78
RowSelection gadget attribute 4–13, C-77

S
Scrolling gadget attribute 4–13, C-79
SelectionMode gadget attribute 4–13, C-80
Sensitive gadget attribute 4–13, C-81
Separator gadget 4–16, C-82
Show gadget attribute 4–13, C-83
Show gadget attributes 4–13
ShowBrowser gadget attribute 4–13, C-84
ShowIndicator gadget attribute 4–13, C-85
Sort gadget attribute 4–13, C-86
state
dialog 3–5
resource 9–15
StatusField dialog attribute 4–13, C-87
StatusMessage dialog attribute 4–13, C-88
String DSL type 4–7
Switch gadget 4–16, C-89

T
Table gadget 4–16, 10–10, C-91
TableColumn gadget C-94
TableRow gadget C-96
tables
cell border style C-12
column width C-24
glyphs C-41
populating C-53
restricting selections C-77, C-80
row height C-76
setting leftmost column C-54
setting number of rows C-78
setting top row C-104
Table gadget C-91
TableColumn gadget C-94
TableRow gadget C-96
TearOff gadget attribute 4–14, C-98
TellModify gadget attribute 4–14, C-99
text fields
encrypting C-36
multi-line C-70
single line C-100
Text gadget C-100
wrapping C-110
Text gadget 4–16, C-100
Title gadget attribute 4–14, C-102
TitlePos gadget attribute 4–14, 4–52, C-103
Tivoli Management Region (TMR)
message catalog directory 4–42
Tivoli Productivity Package 9–2, 9–3
TME desktop. See desktop
TMR. See Tivoli Management Region
TopRow gadget attribute 4–14, C-104
Type gadget attribute 4–14, C-105
V
Value gadget attribute 4–14, C-106
variables
declaring in DSL 4–7
ViewBy gadget attribute 4–14, C-108
Visible gadget attribute 4–14, C-109

W
waddaction command 10–5, 10–9, B–5
waddattr command 6–1, 11–1, B–7
waddpolm command 8–2, 8–3, B–8
waddprop command 10–2, B–10
wconbitmap command 9–2, 9–3, B–11
WD_DISPLAY environment variable 7–2
wdispcnf command 7–17, B–13
wdisperr command 7–19, B–14
wdispmsg command 7–19, B–15
wformatlist command B–16, B–18
wgetaction command 10–6, B–20
wgetattr command 6–2, B–22
wgetbitmap command 9–6, 9–7, 9–19, B–23
wgetdialog command 5–3, 5–4, 5–5, 9–19, B–26
wgetpending command 10–8, B–29
wlsactions command 10–7, B–30, B–32
wlsbitmap command 9–14, B–34
wlsdialog command 5–12, B–36
wlsicon command 9–29, B–38
wputbitmap command 9–8, 9–10, 9–11, 9–22, B–40
wputdialog command 5–7, 5–8, 5–9, 9–22, B–43
wputicon command 9–21, 9–28, B–46
wputmeth command 7–4, 7–9, 7–11, 7–14, B–48
wputstate command 9–23, B–51
Wrap gadget attribute C-110
wrmaction command 10–9, B–52
wrmaattr command 6–3, B–54
wrmbitmap command 9–11, 9–12, 9–13, B–55
wrdialog command 5–9, 5–10, 5–11, B–57
wrmicon command 9–28, B–59
wrmmeth command 7–14, 7–15, 7–16, B–61
wrmpolm command 8–3, B–63
wrmprop command 10–4, B–65
wrunpolm command 8–4, B–66
wsetattr command 6–3, B–68, B–69
wstatusline command 7–25, B–70

X
XPM bitmap format 9–2, B–11