Motif 2.1—Programmer's Reference

Desktop Product Documentation

The Open Group

Desktop Product Documentation:

Motif 2.1—Programmer's Reference, Volume 1
ISBN 1-85912-119-5
Document Number M214A

Motif 2.1—Programmer's Reference, Volume 2
ISBN 1-85912-124-1
Document Number M214B

Motif 2.1—Programmer's Reference, Volume 3
ISBN 1-85912-164-0
Document Number M214C

Published in the U.K. by The Open Group, 1997

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:
OGPubs@opengroup.org

OTHER NOTICES

Contents

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the IT DialTone. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- consolidating, prioritizing, and communicating customer requirements to vendors
- conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- licensing and promoting the Open Brand, represented by the ‘‘X’’ mark, that designates vendor products which conform to Open Group Product Standards
- promoting the benefits of open systems to customers, vendors, and the public.

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

The Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of CAE and Preliminary Specifications through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product. There are currently two forms of Product

Standard, namely the Profile Definition and the Component Definition, although these will eventually be merged into one.

The ‘‘X’’ mark is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the X/Open Trade Mark Licence Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on specification development and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

CAE Specifications

CAE (Common Applications Environment) Specifications are the stable specifications that form the basis for our Product Standards, which are used to develop X/Open branded systems. These specifications are intended to be used widely within the industry for product development and procurement purposes.

Anyone developing products that implement a CAE Specification can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. CAE Specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

Preliminary Specifications

Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as

stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

Preliminary Specifications are analogous to the trial-use standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE Specification. While the intent is to progress Preliminary Specifications to corresponding CAE Specifications, the ability to do so depends on consensus among Open Group members.

Consortium and Technology Specifications

The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as CAE Specifications, in which case the relevant Technology Specification is superseded by a CAE Specification.

In addition, The Open Group publishes:

Product Documentation

This includes product documentation—programmer's guides, user manuals, and so on—relating to the Prestructured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

Guides

These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the CAE Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

Technical Studies

Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as

to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all live documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new Version indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it replaces the previous publication.
- A new Issue indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/public/pubs*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/public/pubs*.

This Book

The *Motif 2.1—Programmer's Reference* contains the reference pages for all Motif programs, Xt widget classes, Xm widget classes, translations, Xm data types and functions, Mrm functions, Uil functions, and file formats.

Audience

This document is written for programmers who want to write applications by using Motif interfaces.

This document assumes that the reader is familiar with the American National Standards Institute (ANSI) C programming language. It also assumes that the reader has a general understanding of the X Window System, the Xlib library, and the X Toolkit Intrinsics (Xt).

Applicability

This is revision 2.1 of this document. It applies to Version 2.1 of the Motif software system.

Purpose

The purpose of this guide is to provide detailed information about all Motif 2.1 programs, widget classes, translations, data types, functions, and file formats for the application developer.

Organization

This document is organized into nine chapter and four appendixes:

- Chapter 1 contains the reference pages for Motif programs.
- Chapter 2 contains the reference pages for Xt widget classes.
- Chapter 3 contains the reference pages for Xm widget classes.
- Chapter 4 contains the reference pages for Motif translations.
- Chapter 5 contains the reference pages for Xm data types.
- Chapter 6 contains the reference pages for Xm functions.
- Chapter 7 contains the reference pages for Mrm functions.
- Chapter 8 contains the reference pages for Uil functions.
- Chapter 9 contains the reference pages for Motif file formats.
- Appendix A contains a list of the constraint arguments and automatically created children for widgets available within UIL (User Interface Language).
- Appendix B contains a list of the reasons and controls, or children, that UIL supports for each Motif Toolkit object.
- Appendix C contains a list of the UIL arguments and their data types.
- Appendix D contains a list of the UIL compiler diagnostics messages.

Reference Page Format

The reference pages in this volume use the following format:

Purpose This section gives a short description of the interface.

Synopsis This section describes the appropriate syntax for using the interface.

Description This section describes the behavior of the interface. On widget reference pages there are tables of resource values in the descriptions. These tables have the following headings:

Name Contains the name of the resource. Each new resource is described following the new resources table.

Class Contains the class of the resource.

Type Contains the type of the resource.

Default Contains the default value of the resource.

Access Contains the access permissions for the resource. A **C** in this column means the resource can be set at widget creation time. An **S** means the resource can be set anytime. A **G** means the resource's value can be retrieved.

Examples This section gives practical examples for using the interface.

Return Values
This section lists the values returned by function interfaces.

Errors/Warnings
This section describes the error conditions associated with using this interface.

Related Information
This section provides cross-references to related interfaces and header files described within this document.

Related Documents

For information on Motif and CDE style, refer to the following documents:

CDE 2.1/Motif 2.1—Style Guide and Glossary
Document Number M027 ISBN 1-85912-104-7

CDE 2.1/Motif 2.1—Style Guide Certification Checklist
Document Number M028 ISBN 1-85912-109-8

CDE 2.1/Motif 2.1—Style Guide Reference
Document Number M029 ISBN 1-85912-114-4

For additional information about Motif and CDE, refer to the following Desktop Documentation:

CDE 2.1/Motif 2.1—User's Guide
Document Number M021 ISBN 1-85912-173-X

CDE 2.1—System Manager's Guide
Document Number M022 ISBN 1-85912-178-0

CDE 2.1—Programmer's Overview and Guide
Document Number M023 ISBN 1-85912-183-7

CDE 2.1—Programmer's Reference, Volume 1
Document Number M024A ISBN 1-85912-188-8

CDE 2.1—Programmer's Reference, Volume 2
Document Number M024B ISBN 1-85912-193-4

CDE 2.1—Programmer's Reference, Volume 3
Document Number M024C ISBN 1-85912-174-8

CDE 2.1—Application Developer's Guide
Document Number M026 ISBN 1-85912-198-5

Motif 2.1—Programmer's Guide
Document Number M213 ISBN 1-85912-134-9

Motif 2.1—Widget Writer's Guide
Document Number M216 ISBN 1-85912-129-2

For additional information about Xlib and Xt, refer to the following X Window System documents:

Xlib—C Language X Interface

X Toolkit Intrinsics—C Language Interface

Typographic and Keying Conventions

This book uses the following conventions.

DocBook SGML Conventions

This book is written in the Structured Generalized Markup Language (SGML) using the DocBook Document Type Definition (DTD). The following table describes the DocBook markup used for various semantic elements.

Markup Appearance	Semantic Element(s)	Example
AaBbCc123	The names of commands.	Use the **ls** command to list files.
AaBbCc123	The names of command options.	Use **ls –a** to list all files.
AaBbCc123	Command-line placeholder: replace with a real name or value.	To delete a file, type **rm** *filename*.
AaBbCc123	The names of files and directories.	Edit your **.login** file.
AaBbCc123	Book titles, new words or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Terminology Conventions

Components of the user interface are represented by uppercase letters for each major word in the name of the component, such as PushButton. In addition, this book uses the term *primitive* to mean any subclass of **XmPrimitive** and the term *manager* to mean any subclass of **XmManager**. Note that both of these terms are in lowercase.

Keyboard Conventions

Because not all keyboards are the same, it is difficult to specify keys that are correct for every manufacturer's keyboard. To solve this problem, this guide describes keys that use a *virtual key* mechanism. The term *virtual* implies that the keys as described do not necessarily correspond to a fixed set of actual keys. Instead, virtual keys are

linked to actual keys by means of *virtual bindings*. A given virtual key may be bound to different physical keys for different keyboards.

See Chapter 13 of the *Motif 2.1—Programmer's Guide* for information on the mechanism for binding virtual keys to actual keys. For details, see the **VirtualBindings**(3) reference page in this manual.

Mouse Conventions

Mouse buttons are described in this reference by using a **virtual button** mechanism to better describe behavior independent from the number of buttons on the mouse. This guide assumes a 3-button mouse. On a 3-button mouse, the leftmost mouse button is usually defined as **BSelect**, the middle mouse button is usually defined as **BTransfer**, and the rightmost mouse button is usually defined as **BMenu**. For details about how virtual mouse buttons are usually defined, see the **VirtualBindings**(3) reference page in this document.

Problem Reporting

If you have any problems with the software or vendor-supplied documentation, contact your software vendor's customer service department. Comments relating to this Open Group document, however, should be sent to the addresses provided on the copyright page.

Trademarks

UnixWare is a trademark of Novell, Inc.

Microsoft Windows is a trademark of Microsoft.

OS/2 is a trademark of International Business Machines Corp.

X Window System is a trademark of X Consortium, Inc.

Chapter 7

Mrm Functions

MrmCloseHierarchy

Purpose

Closes a UID hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmCloseHierarchy(
        MrmHierarchy hierarchy_id);
```

Description

The **MrmCloseHierarchy** function closes a UID hierarchy previously opened by **MrmOpenHierarchyPerDisplay**. All files associated with the hierarchy are closed by the Motif Resource Manager (MRM) and all associated memory is returned.

hierarchy_id Specifies the ID of a previously opened UID hierarchy. The *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmFAILURE
The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3).

MrmFetchBitmapLiteral

Purpose

Fetches a bitmap literal from a hierarchy

Synopsis

#include <Mrm/MrmPublic.h>

```
Cardinal MrmFetchBitmapLiteral(
        MrmHierarchy hierarchy_id,
        String index,
        Screen *screen,
        Display *display,
        Pixmap *pixmap_return,
        Dimension *width,
        Dimension *height);
```

Description

The **MrmFetchBitmapLiteral** function fetches a bitmap literal from an MRM hierarchy, and converts the bitmap literal to an X pixmap of depth 1. The function returns this pixmap and its width and height.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified icon literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the bitmap literal to fetch.

screen Specifies the screen used for the pixmap. The *screen* argument specifies a pointer to the Xlib structure **Screen** which contains the information about that screen and is linked to the **Display** structure. For more information on the **Display** and **Screen** structures, see the Xlib function **XOpenDisplay** and the associated screen information macros.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

pixmap_return
: Returns the resulting X pixmap value. The function allocates space for this pixmap. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmDestroyPixmap**.

width
: Specifies a pointer to the width of the pixmap.

height
: Specifies a pointer to the height of the pixmap.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
: The function executed successfully.

MrmBAD_HIERARCHY
: The hierarchy ID was invalid.

MrmNOT_FOUND
: The bitmap literal was not found in the hierarchy.

MrmWRONG_TYPE
: The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE
: The function failed.

Related Information

MrmFetchIconLiteral(3), **MrmFetchLiteral**(3), and **XOpenDisplay**(3).

MrmFetchColorLiteral

Purpose

Fetches a named color literal from a UID file

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchColorLiteral(
MrmHierarchy *hierarchy_id***,**
String *index***,**
Display *****display*,**
Colormap *colormap_id***,**
Pixel *****pixel*);**

Description

The **MrmFetchColorLiteral** function fetches a named color literal from a UID file, and converts the color literal to a pixel color value.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the color literal to fetch. You must define this name in UIL as an exported value.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

colormap_id Specifies the ID of the color map. If *colormap_id* is NULL, the default color map is used.

pixel Returns the ID of the color literal.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND
The color literal was not found in the UIL file.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE
The function failed.

Related Information

MrmFetchBitmapLiteral(3), **MrmOpenHierarchyPerDisplay**(3), **MrmFetchIconLiteral**(3), **MrmFetchLiteral**(3), and **XOpenDisplay**(3).

MrmFetchIconLiteral

Purpose

Fetches an icon literal from a hierarchy

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchIconLiteral(
MrmHierarchy *hierarchy_id***,**
String *index***,**
Screen ****screen*,**
Display ****display*,**
Pixel *fgpix***,**
Pixel *bgpix***,**
Pixmap ****pixmap*);**

Description

The **MrmFetchIconLiteral** function fetches an icon literal from an MRM hierarchy and converts the icon literal to an X pixmap.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified icon literal. The *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the icon literal to fetch.

screen Specifies the screen used for the pixmap. The *screen* argument specifies a pointer to the Xlib structure **Screen**, which contains the information about that screen and is linked to the **Display** structure. For more information on the **Display** and **Screen** structures, see the Xlib function **XOpenDisplay** and the associated screen information macros.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

fgpix Specifies the foreground color for the pixmap.

bgpix Specifies the background color for the pixmap.

pixmap Returns the resulting X pixmap value. The function allocates space for this pixmap. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmDestroyPixmap**.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND
The icon literal was not found in the hierarchy.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE
The function failed.

Related Information

MrmFetchBitmapLiteral(3), **MrmOpenHierarchyPerDisplay**(3), **MrmFetchLiteral**(3), **MrmFetchColorLiteral**(3), and **XOpenDisplay**(3).

MrmFetchLiteral

Purpose

Fetches a literal from a UID file

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchLiteral(
MrmHierarchy *hierarchy_id***,**
String *index***,**
Display *****display*,**
XtPointer *****value*,**
MrmCode *****type*);**

Description

The **MrmFetchLiteral** function reads and returns the value and type of a literal (named value) that is stored as a public resource in a single UID file. This function returns a pointer to the value of the literal. For example, an integer is always returned as a pointer to an integer, and a string is always returned as a pointer to a string.

Applications should not use **MrmFetchLiteral** for fetching icon or color literals. If this is attempted, **MrmFetchLiteral** returns an error.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the literal (pixmap) to fetch. You must define this name in UIL as an exported value.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

value Returns the ID of the named literal's value. The function allocates space for the returned value. The application is responsible for managing the allocated space by calling the appropriate deallocation function. For

example, if the returned ID symbolizes a pixmap, then the application can recover the allocated space by calling **XmDestroyPixmap**.

type Returns the named literal's data type. Types are defined in the include file **Mrm/MrmPublic.h**.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND
The literal was not found in the UIL file.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE
The function failed.

Related Information

MrmFetchBitmapLiteral(3), **MrmOpenHierarchyPerDisplay**(3), **MrmFetchIconLiteral**(3), **MrmFetchColorLiteral**(3), and **XOpenDisplay**(3).

MrmFetchSetValues

Purpose

Fetches the values to be set from literals stored in UID files

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchSetValues(
MrmHierarchy *hierarchy_id***,**
Widget *widget***,**
ArgList *args***,**
Cardinal *num_args***);**

Description

The **MrmFetchSetValues** function is similar to **XtSetValues**, except that the values to be set are defined by the UIL named values that are stored in the UID hierarchy. **MrmFetchSetValues** fetches the values to be set from literals stored in UID files.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

widget Specifies the widget that is modified.

args Specifies an argument list that identifies the widget arguments to be modified as well as the index (UIL name) of the literal that defines the value for that argument. The name part of each argument (*args[n].name*) must begin with the string **XmN** followed by the name that uniquely identifies this attribute tag. For example, **XmNwidth** is the attribute name associated with the core argument *width*. The value part (*args[n].value*) must be a string that gives the index (UIL name) of the literal. You must define all literals in UIL as exported values.

num_args Specifies the number of entries in *args*.

This function sets the values on a widget, evaluating the values as public literal resource references resolvable from a UID hierarchy. Each literal is fetched from the

hierarchy, and its value is modified and converted as required. This value is then placed in the argument list and used as the actual value for an **XtSetValues** call. **MrmFetchSetValues** allows a widget to be modified after creation using UID file values the same way creation values are used in **MrmFetchWidget**.

As in **MrmFetchWidget**, each argument whose value can be evaluated from the UID hierarchy is set in the widget. Values that are not found or values in which conversion errors occur are not modified.

Each entry in the argument list identifies an argument to be modified in the widget. The name part identifies the tag, which begins with **XmN**. The value part must be a string whose value is the index of the literal. Thus, the following code would modify the label resource of the widget to have the value of the literal accessed by the index **OK_button_label** in the hierarchy:

```
args[n].name = XmNlabel;
args[n].value = "OK_button_label";
```

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmPARTIAL_SUCCESS
At least one literal was successfully fetched.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmFAILURE
The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3), **XtSetValues**(3).

MrmFetchWidget

Purpose

Fetches and creates an indexed (UIL named) application widget and its children

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchWidget(
MrmHierarchy *hierarchy_id***,**
String *index***,**
Widget *parent_widget***,**
Widget ****widget*,**
MrmType ****class*);**

Description

The **MrmFetchWidget** function fetches and creates an indexed application widget and its children. The indexed application widget is any widget that is named in UIL. In fetch operations, the fetched widget's subtree is also fetched and created. This widget must not appear as the child of a widget within its own subtree. **MrmFetchWidget** does not execute **XtManageChild** for the newly created widget.

All widgets fetched by a call to **MrmFetchWidget** are not managed at the time of their creation callbacks.

hierarchy_id Specifies the ID of the *UID* hierarchy that contains the interface definition. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the widget to fetch.

parent_widget
Specifies the parent widget ID.

widget Returns the widget ID of the created widget.

class This argument must be set to an actual pointer; it cannot be a NULL pointer. **MrmFetchWidget** sets this argument to an implementation dependent value.

An application can fetch any named widget in the *UID* hierarchy using **MrmFetchWidget**. **MrmFetchWidget** can be called at any time to fetch a widget that was not fetched at application startup. **MrmFetchWidget** can be used to defer fetching pop-up widgets until they are first referenced (presumably in a callback), and then used to fetch them once.

MrmFetchWidget can also create multiple instances of a widget (and its subtree). In this case, the *UID* definition functions as a template; a widget definition can be fetched any number of times. An application can use this template to make multiple instances of a widget, for example, in a dialog box box or menu.

The index (UIL name) that identifies the widget must be known to the application.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND
The widget was not found in UID hierarchy.

MrmFAILURE
The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3), **MrmFetchWidgetOverride**(3).

MrmFetchWidgetOverride

Purpose

Fetches any indexed (UIL named) application widget. It overrides the arguments specified for this application widget in UIL

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchWidgetOverride(
MrmHierarchy *hierarchy_id***,**
String *index***,**
Widget *parent_widget***,**
String *override_name***,**
ArgList *override_args***,**
Cardinal *override_num_args***,**
Widget ***_widget_,**
MrmType ***_class_);**

Description

The **MrmFetchWidgetOverride** function is the extended version of **MrmFetchWidget**. It is identical to **MrmFetchWidget**, except that it allows the caller to override the widget's name and any arguments that **MrmFetchWidget** would otherwise retrieve from the UID file or one of the defaulting mechanisms. That is, the override argument list is not limited to those arguments in the UID file.

The override arguments apply only to the widget fetched and returned by this function. Its children (subtree) do not receive any override parameters.

hierarchy_id Specifies the ID of the UID hierarchy that contains the interface definition. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the widget to fetch.

parent_widget
Specifies the parent widget ID.

override_name
Specifies the name to override the widget name. Use a NULL value if you do not want to override the widget name.

override_args
Specifies the override argument list, exactly as given to **XtCreateWidget** (conversion complete and so forth). Use a NULL value if you do not want to override the argument list.

override_num_args
Specifies the number of arguments in *override_args*.

widget Returns the widget ID of the created widget.

class Returns the class code identifying MRM's widget class. Literals identifying MRM widget class codes are defined in the include file **Mrm/MrmPublic.h**.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND
The widget was not found in UID hierarchy.

MrmFAILURE
The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3), **MrmFetchWidget**(3).

MrmInitialize

Purpose Prepares an application to use MRM widget-fetching facilities

Synopsis

```
void MrmInitialize(
        Void);
```

Description

The **MrmInitialize** function must be called to prepare an application to use MRM widget-fetching facilities. You must call this function prior to fetching a widget. However, it is good programming practice to call **MrmInitialize** prior to performing any MRM operations.

MrmInitialize initializes the internal data structures that MRM needs to successfully perform type conversion on arguments and to successfully access widget creation facilities. An application must call **MrmInitialize** before it uses other MRM functions.

MrmOpenHierarchy

Purpose

Allocates a hierarchy ID and opens all the UID files in the hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchy(
        MrmCount num_files,
        String file_names_list[],
        MrmOsOpenParamPtr *ancillary_structures_list,
        MrmHierarchy *hierarchy_id);
```

Description

This routine is obsolete and exists for compatibility with previous releases. It is replaced by **MrmOpenHierarchyPerDisplay**. **MrmOpenHierarchy** is identical to **MrmOpenHierarchyPerDisplay** except that **MrmOpenHierarchy** does not take a *display* argument.

num_files Specifies the number of files in the name list.

file_names_list
Specifies an array of character strings that identify the UID files.

ancillary_structures_list
A list of operating-system-dependent ancillary structures corresponding to items such as filenames, clobber flags, and so forth. This argument should be NULL for most operations. If you need to reference this structure, see the definition of **MrmOsOpenParamPtr** in the **MrmPublic.h** header file for more information.

hierarchy_id Returns the search hierarchy ID. The search hierarchy ID identifies the list of UID files that MRM searches (in order) when performing subsequent fetch calls.

Each UID file string in *file_names_list* can specify either a full pathname or a filename. If a UID file string has a leading slash (/), it specifies a full pathname, and MRM opens

the file as specified. Otherwise, the UID file string specifies a filename. In this case, MRM looks for the file along a search path specified by the **UIDPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set.

The **UIDPATH** environment variable specifies a search path and naming conventions associated with UID files. It can contain the substitution field **%U**, where the UID file string from the *file_names_list* argument to **MrmOpenHierarchyPerDisplay** is substituted for **%U**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to *uid*. The entire path is first searched with **%S** mapped to **.uid**. If no file is found, it is searched again with **%S** mapped to NULL.

If no display is set prior to calling this function, the result of this function's call to **XtResolvePathname** is undefined.

For example, the following **UIDPATH** value and **MrmOpenHierarchy** call cause MRM to open two separate UID files:

```
UIDPATH=/uidlib/%L/%U.uid:/uidlib/%U/%L
  static char *uid_files[] = {"/usr/users/me/test.uid", "test2"};
  MrmHierarchy  *Hierarchy_id;
  MrmOpenHierarchy((MrmCount)2,uid_files, NULL, Hierarchy_id)
```

MRM opens the first file, **/usr/users/me/test.uid**, as specified in the *file_names_list* argument to **MrmOpenHierarchy**, because the UID file string in the *file_names_list* argument specifies a full pathname. MRM looks for the second file, **test2**, first as **/uidlib/%L/test2.uid** and second as **/uidlib/test2/%L**, where the display's language string is substituted for **%L**.

After **MrmOpenHierarchy** opens the UID hierarchy, you should not delete or modify the UID files until you close the UID hierarchy by calling **MrmCloseHierarchy**.

If **UIDPATH** is not set but the environment variable **XAPPLRESDIR** is set, MRM searches the following pathnames:

- **%U%S**
- **$XAPPLRESDIR/%L/uid/%N/%U%S**
- **$XAPPLRESDIR/%l/uid/%N/%U%S**
- **$XAPPLRESDIR/uid/%N/%U%S**
- **$XAPPLRESDIR/%L/uid/%U%S**

- **$XAPPLRESDIR/%l/uid/%U%S**
- **$XAPPLRESDIR/uid/%U%S**
- **$HOME/uid/%U%S**
- **$HOME/%U%S**
- **/usr/lib/X11/%L/uid/%N/%U%S**
- **/usr/lib/X11/%l/uid/%N/%U%S**
- **/usr/lib/X11/uid/%N/%U%S**
- **/usr/lib/X11/%L/uid/%U%S**
- **/usr/lib/X11/%l/uid/%U%S**
- **/usr/lib/X11/uid/%U%S**
- **/usr/include/X11/uid/%U%S**

If neither **UIDPATH** nor **XAPPLRESDIR** is set, MRM searches the following pathnames:

- **%U%S**
- **HOME/%L/uid/%N/%U%S**
- **HOME/%l/uid/%N/%U%S**
- **$HOME/uid/%N/%U%S**
- **$HOME/%L/uid/%U%S**
- **$HOME/%l/uid/%U%S**
- **$HOME/uid/%U%S**
- **$HOME/%U%S**
- **/usr/lib/X11/%L/uid/%N/%U%S**
- **/usr/lib/X11/%l/uid/%N/%U%S**
- **/usr/lib/X11/uid/%N/%U%S**
- **/usr/lib/X11/%L/uid/%U%S**
- **/usr/lib/X11/%l/uid/%U%S**
- **/usr/lib/X11/uid/%U%S**

- **/usr/include/X11/uid/%U%S**

These paths are defaults that vendors may change. For example, a vendor may use different directories for **/usr/lib/X11** and **/usr/include/X11**.

The following substitutions are used in these paths:

%U The UID file string, from the *file_names_list* argument.

%N The class name of the application.

%L The display's language string. This string is influenced by **XtSetLanguageProc**. The default string is determined by calling setlocale(*LC_ALL*, *NULL*).

%l The language component of the display's language string.

%S The suffix to the filename. The entire path is first searched with a suffix of **.uid**. If no file is found, it is searched again with a NULL suffix.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmNOT_FOUND
File not found.

MrmFAILURE
The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3) and **MrmCloseHierarchy**(3).

MrmOpenHierarchyFromBuffer

Purpose

Allocates a hierarchy ID and opens a buffer containing a memory image of a UID file

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchyFromBuffer(
        unsigned char uid_buffer,
        MrmHierarchy *hierarchy_id);
```

Description

MrmOpenHierarchyFromBuffer allows you to specify a buffer containing information from UID files that MRM searches in subsequent fetch operations. This function also allocates a hierarchy ID and initializes the optimized search lists in the hierarchy.

buffer Specifies a stream of bytes containing information from UID files

hierarchy_id Returns the search hierarchy ID. The search hierarchy ID identifies the buffer that MRM searches when performing subsequent fetch calls.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmFAILURE
The function failed.

Related Information

MrmCloseHierarchy(3) and **MrmOpenHierarchyPerDisplay**(3).

MrmOpenHierarchyPerDisplay

Purpose

Allocates a hierarchy ID and opens all the UID files in the hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchyPerDisplay(
        Display *display,
        MrmCount num_files,
        String file_names_list[],
        MrmOsOpenParamPtr *ancillary_structures_list,
        MrmHierarchy *hierarchy_id);
```

Description

MrmOpenHierarchyPerDisplay allows you to specify the list of UID files that MRM searches in subsequent fetch operations. All subsequent fetch operations return the first occurrence of the named item encountered while traversing the UID hierarchy from the first list element (UID file specification) to the last list element. This function also allocates a hierarchy ID and opens all the UID files in the hierarchy. It initializes the optimized search lists in the hierarchy. If **MrmOpenHierarchyPerDisplay** encounters any errors during its execution, any files that were opened are closed.

The application must call **XtAppInitialize** before calling **MrmOpenHierarchyPerDisplay**.

display Specifies the connection to the X server and the value to pass to **XtResolvePathname**. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

num_files Specifies the number of files in the name list.

file_names_list
Specifies an array of character strings that identify the UID files.

ancillary_structures_list

A list of operating-system-dependent ancillary structures corresponding to items such as filenames, clobber flags, and so forth. This argument should be NULL for most operations. If you need to reference this structure, see the definition of **MrmOsOpenParamPtr** in the **MrmPublic.h** header file for more information.

hierarchy_id Returns the search hierarchy ID. The search hierarchy ID identifies the list of UID files that MRM searches (in order) when performing subsequent fetch calls.

Each UID file string in *file_names_list* can specify either a full pathname or a filename. If a UID file string has a leading / (slash), it specifies a full pathname, and MRM opens the file as specified. Otherwise, the UID file string specifies a filename. In this case MRM looks for the file along a search path specified by the **UIDPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set.

The **UIDPATH** environment variable specifies a search path and naming conventions associated with UID files. It can contain the substitution field **%U**, where the UID file string from the *file_names_list* argument to **MrmOpenHierarchyPerDisplay** is substituted for **%U**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to *uid*. The entire path is searched first with **%S** mapped to **.uid**. If no file is found, it is searched again with **%S** mapped to NULL. For example, the following **UIDPATH** value and **MrmOpenHierarchyPerDisplay** call cause MRM to open two separate UID files:

```
UIDPATH=/uidlib/%L/%U.uid:/uidlib/%U/%L
  static char *uid_files[] = {"/usr/users/me/test.uid", "test2"};
  MrmHierarchy  *Hierarchy_id;
  MrmOpenHierarchyPerDisplay((MrmCount)2,uid_files, NULL, Hierarchy_id)
```

MRM opens the first file, **/usr/users/me/test.uid**, as specified in the *file_names_list* argument to **MrmOpenHierarchyPerDisplay**, because the UID file string in the *file_names_list* argument specifies a full pathname. MRM looks for the second file, **test2**, first as **/uidlib/%L/test2.uid** and second as **/uidlib/test2/%L**, where the display's language string is substituted for **%L**.

After **MrmOpenHierarchyPerDisplay** opens the UID hierarchy, you should not delete or modify the UID files until you close the UID hierarchy by calling **MrmCloseHierarchy**.

If **UIDPATH** is not set, but the environment variable **XAPPLRESDIR** is set, MRM searches the following pathnames:

- **%U%S**
- **$XAPPLRESDIR/%L/uid/%N/%U%S**
- **$XAPPLRESDIR/%l/uid/%N/%U%S**
- **$XAPPLRESDIR/uid/%N/%U%S**
- **$XAPPLRESDIR/%L/uid/%U%S**
- **$XAPPLRESDIR/%l/uid/%U%S**
- **$XAPPLRESDIR/uid/%U%S**
- **$HOME/uid/%U%S**
- **$HOME/%U%S**
- **/usr/lib/X11/%L/uid/%N/%U%S**
- **/usr/lib/X11/%l/uid/%N/%U%S**
- **/usr/lib/X11/uid/%N/%U%S**
- **/usr/lib/X11/%L/uid/%U%S**
- **/usr/lib/X11/%l/uid/%U%S**
- **/usr/lib/X11/uid/%U%S**
- **/usr/include/X11/uid/%U%S**

If neither **UIDPATH** nor **XAPPLRESDIR** is set, MRM searches the following pathnames:

- **%U%S**
- **$HOME/%L/uid/%N/%U%S**
- **$HOME/%l/uid/%N/%U%S**
- **$HOME/uid/%N/%U%S**
- **$HOME/%L/uid/%U%S**
- **$HOME/%l/uid/%U%S**
- **$HOME/uid/%U%S**

- **$HOME/%U%S**
- **/usr/lib/X11/%L/uid/%N/%U%S**
- **/usr/lib/X11/%l/uid/%N/%U%S**
- **/usr/lib/X11/uid/%N/%U%S**
- **/usr/lib/X11/%L/uid/%U%S**
- **/usr/lib/X11/%l/uid/%U%S**
- **/usr/lib/X11/uid/%U%S**
- **/usr/include/X11/uid/%U%S**

These paths are defaults that vendors may change. For example, a vendor may use different directories for **/usr/lib/X11** and **/usr/include/X11**.

The following substitutions are used in these paths:

%U The UID file string, from the *file_names_list* argument.

%N The class name of the application.

%L The display's language string. This string is influenced by **XtSetLanguageProc**. The default string is determined by calling setlocale(*LC_ALL, NULL*).

%l The language component of the display's language string.

%S The suffix to the filename. The entire path is first searched with a suffix of **.uid**. If no file is found, it is searched again with a NULL suffix.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmNOT_FOUND
File not found.

MrmFAILURE
The function failed.

Related Information

MrmCloseHierarchy(3).

MrmRegisterClass

Purpose

Saves the information needed for MRM to access the widget creation function for user-defined widgets

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterClass(
MrmType *class_code***,**
String *class_name***,**
String *create_name***,**
Widget (*create_proc**) (),**
WidgetClass *class_record***);**

Description

The **MrmRegisterClass** function allows MRM to access user-defined widget classes. This function registers the necessary information for MRM to create widgets of this class. You must call **MrmRegisterClass** prior to fetching any user-defined class widget.

MrmRegisterClass saves the information needed to access the widget creation function and to do type conversion of argument lists by using the information in MRM databases.

class_code This argument is ignored; it is present for compatibility with previous releases.

class_name This argument is ignored; it is present for compatibility with previous releases.

create_name Specifies the case-sensitive name of the low-level widget creation function for the class. An example from the Motif Toolkit is **XmCreateLabel**. Arguments are *parent_widget*, *name*, *override_arglist*, and *override_argcount*.

For user-defined widgets, *create_name* is the creation procedure in the UIL that defines this widget.

create_proc Specifies the address of the creation function that you named in *create_name*.

class_record Specifies a pointer to the class record.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmFAILURE
The function failed.

MrmRegisterNames

Purpose

Registers the values associated with the names referenced in UIL (for example, UIL callback function names or UIL identifier names)

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterNames(
MrmRegisterArglist *register_list***,**
MrmCount *register_count***);**

Description

The **MrmRegisterNames** function registers a vector of names and associated values for access in MRM. The values can be callback functions, pointers to user-defined data, or any other values. The information provided is used to resolve symbolic references occurring in UID files to their run-time values. For callbacks, this information provides the procedure address required by the Motif Toolkit. For names used as identifiers in UIL, this information provides any run-time mapping the application needs.

This function is similar to **MrmRegisterNamesInHierarchy**, except that the scope of the names registered by **MrmRegisterNamesInHierarchy** is limited to the hierarchy specified in the call to that function, whereas the names registered by **MrmRegisterNames** have global scope. When MRM looks up a name, it first tries to find the name among those registered for the given hierarchy. If that lookup fails, it tries to find the name among those registered globally.

register_list Specifies a list of name/value pairs for the names to be registered. Each name is a case-sensitive, NULL-terminated ASCII string. Each value is a 32-bit quantity, interpreted as a procedure address if the name is a callback function, and uninterpreted otherwise.

register_count
Specifies the number of entries in *register_list*.

The names in the list are case-sensitive. The list can be either ordered or unordered.

Callback functions registered through **MrmRegisterNames** can be either regular or creation callbacks. Regular callbacks have declarations determined by Motif Toolkit and user requirements. Creation callbacks have the same format as any other callback:

void CallBackProc(
Widget ****widget_id*,**
Opaque *tag***,**
XmAnyCallbackStruct ****callback_data*);**

widget_id Specifies the widget ID associated with the widget performing the callback (as in any callback function).

tag Specifies the tag value (as in any callback function).

callback_data
Specifies a widget-specific data structure. This data structure has a minimum of two members: event and reason. The reason member is always set to **MrmCR_CREATE**.

Note that the widget name and parent are available from the widget record accessible through *widget_id*.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmFAILURE
The function failed.

MrmRegisterNamesInHierarchy

Purpose

Registers the values associated with the names referenced in UIL within a single hierarchy (for example, UIL callback function names or UIL identifier names)

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterNamesInHierarchy(
MrmHierarchy *hierarchy_id***,**
MrmRegisterArglist *register_list***,**
MrmCount *register_count***);**

Description

The **MrmRegisterNamesInHierarchy** function registers a vector of names and associated values for access in MRM. The values can be callback functions, pointers to user-defined data, or any other values. The information provided is used to resolve symbolic references occurring in UID files to their run-time values. For callbacks, this information provides the procedure address required by the Motif Toolkit. For names used as identifiers in UIL, this information provides any run-time mapping the application needs.

This function is similar to **MrmRegisterNames**, except that the scope of the names registered by **MrmRegisterNamesInHierarchy** is limited to the hierarchy specified by *hierarchy_id*, whereas the names registered by **MrmRegisterNames** have global scope. When MRM looks up a name, it first tries to find the name among those registered for the given hierarchy. If that lookup fails, it tries to find the name among those registered globally.

hierarchy_id Specifies the hierarchy with which the names are to be associated.

register_list Specifies a list of name/value pairs for the names to be registered. Each name is a case-sensitive, NULL-terminated ASCII string. Each value is a 32-bit quantity, interpreted as a procedure address if the name is a callback function, and uninterpreted otherwise.

register_count
Specifies the number of entries in *register_list*.

The names in the list are case-sensitive. The list can be either ordered or unordered.

Callback functions registered through **MrmRegisterNamesInHierarchy** can be either regular or creation callbacks. Regular callbacks have declarations determined by Motif Toolkit and user requirements. Creation callbacks have the same format as any other callback:

void CallBackProc(
Widget ****widget_id*,**
Opaque *tag***,**
XmAnyCallbackStruct ****callback_data*);**

widget_id Specifies the widget ID associated with the widget performing the callback (as in any callback function).

tag Specifies the tag value (as in any callback function).

callback_data
Specifies a widget-specific data structure. This data structure has a minimum of two members: event and reason. The reason member is always set to **MrmCR_CREATE**.

Note that the widget name and parent are available from the widget record accessible through *widget_id*.

Return Values

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmFAILURE
The function failed.

Chapter 8

Uil Functions

Uil

Purpose

Invokes the UIL compiler from within an application

Synopsis

```
#include <uil/UilDef.h>

Uil_status_type Uil(
        Uil_command_type *command_desc,
        Uil_compile_desc_type **compile_desc,
        Uil_continue_type (*message_cb) (),
        char *message_data,
        Uil_continue_type (*status_cb) (),
        char *status_data);
```

Description

The **Uil** function provides a callable entry point for the UIL compiler. The **Uil** callable interface can be used to process a UIL source file and to generate UID files, as well as return a detailed description of the UIL source module in the form of a symbol table (parse tree).

command_desc
: Specifies the **uil** command line.

compile_desc
: Returns the results of the compilation.

message_cb
: Specifies a callback function that is called when the compiler encounters errors in the UIL source.

message_data
: Specifies user data that is passed to the message callback function (*message_cb)*. Note that this argument is not interpreted by UIL, and is used exclusively by the calling application.

status_cb Specifies a callback function that is called to allow X applications to service X events such as updating the screen. This function is called at various check points, which have been hard coded into the UIL compiler. The *status_update_delay* argument in *command_desc* specifies the number of check points to be passed before the *status_cb* function is invoked.

status_data Specifies user data that is passed to the status callback function (*status_cb).* Note that this argument is not interpreted by the UIL compiler and is used exclusively by the calling application.

Following are the data structures *Uil_command_type* and *Uil_compile_desc_type*:

```
typedef struct Uil_command_type {
char *source_file;
    /* single source to compile */
char *resource_file; /* name of output file */
char *listing_file; /* name of listing file */
unsigned int *include_dir_count;
    /* number of dirs. in include_dir */
char *((*include_dir) []);
    /* dir. to search for include files */
unsigned listing_file_flag: 1;
    /* produce a listing */
unsigned resource_file_flag: 1;
    /* generate UID output */
unsigned machine_code_flag: 1;
    /* generate machine code */
unsigned report_info_msg_flag: 1;
    /* report info messages */
unsigned report_warn_msg_flag: 1;
    /* report warnings */
unsigned parse_tree_flag: 1;
    /* generate parse tree */
unsigned int status_update_delay;
    /* number of times a status point is */
    /* passed before calling status_cb */
    /* function 0 means called every time */
char *database;
    /* name of database file */
unsigned database_flag: 1;
```

```
    /* read a new database file */
unsigned use_setlocale_flag: 1;
    /* enable calls to setlocale */
};
typedef struct Uil_compile_desc_type {
unsigned int compiler_version;
    /* version number of compiler */
unsigned int data_version;
    /* version number of structures */
char *parse_tree_root; /* parse tree output */
unsigned int message_count [Uil_k_max_status+1];
/* array of severity counts */
};
```

Following is a description of the message callback function specified by *message_cb*:

Uil_continue_type (**message_cb*) (*message_data, message_number, severity, msg_buffer, src_buffer, ptr_buffer, loc_buffer, message_count*)
char **message_data*;
int *message_number*;
int *severity*;
char **msg_buffer*, **src_buffer*;
char **ptr_buffer*, **loc_buffer*;
int *message_count*[];

This function specifies a callback function that UIL invokes instead of printing an error message when the compiler encounters an error in the UIL source. The callback should return one of the following values:

Uil_k_terminate
Terminate processing of the source file

Uil_k_continue
Continue processing the source file

The arguments are

message_data
Data supplied by the application as the *message_data* argument to the **Uil** function. UIL does not interpret this data in any way; it just passes it to the callback.

message_number
An index into a table of error messages and severities for internal use by UIL.

severity An integer that indicates the severity of the error. The possible values are the status constants returned by the **Uil** function. See **Return Value** for more information.

msg_buffer A string that describes the error.

src_buffer A string consisting of the source line where the error occurred. This string is not always available. In this case, the argument is NULL.

ptr_buffer A string consisting of whitespace and a printing character in the character position corresponding to the column of the source line where the error occurred. This string may be printed beneath the source line to provide a visual indication of the column where the error occurred. This string is not always available. In this case, the argument is NULL.

loc_buffer A string identifying the line number and file of the source line where the error occurred. This is not always available; the argument is then NULL.

message_count
An array of integers containing the number of diagnostic messages issued thus far for each severity level. To find the number of messages issued for the current severity level, use the *severity* argument as the index into this array.

Following is a description of the status callback function specified by *status_cb*:

```
Uil_continue_type (*status_cb) (status_data, percent_complete,
    lines_processed, current_file, message_count)
     char *status_data;
     int percent_complete;
     int lines_processed;
     char *current_file;
     int message_count[];
```

This function specifies a callback function that is invoked to allow X applications to service X events such as updating the screen. The callback should return one of the following values:

Uil_k_terminate
Terminate processing of the source file

Uil_k_continue
Continue processing the source file

The arguments are

status_data Data supplied by the application as the *status_data* argument to the **Uil** function. UIL does not interpret this data in any way; it just passes it to the callback.

percent_complete
An integer indicating what percentage of the current source file has been processed so far.

lines_processed
An integer indicating how many lines of the current source file have been read so far.

current_file A string containing the pathname of the current source file.

message_count
An array of integers containing the number of diagnostic messages issued thus far for each severity level. To find the number of messages issued for a given severity level, use the severity level as the index into this array. The possible severity levels are the status constants returned by the **Uil** function. See **Return Value** for more information.

Return Values

This function returns one of the following status return constants:

Uil_k_success_status
The operation succeeded.

Uil_k_info_status
The operation succeeded. An informational message is returned.

Uil_k_warning_status
The operation succeeded. A warning message is returned.

Uil_k_error_status
The operation failed due to an error.

Uil_k_severe_status
The operation failed due to an error.

Related Information

UilDumpSymbolTable(3) and **uil**(1).

UilDumpSymbolTable

Purpose

Dumps the contents of a named UIL symbol table to standard output

Synopsis

#include <uil/UilDef.h>

void UilDumpSymbolTable(
sym_entry_type ****root_ptr*);**

Description

The **UilDumpSymbolTable** function dumps the contents of a UIL symbol table pointer to standard output.

root_ptr Specifies a pointer to the the symbol table root entry. This value can be taken from the **parse_tree_root** part of the **Uil_compile_desc_type** data structure returned by **Uil**.

By following the link from the root entry, you can traverse the entire parse tree. Symbol table entries are in the following format:

hex.address symbol.type symbol.data prev.source.position source.position modification.record

where:

hex.address Specifies the hexadecimal address of this entry in the symbol table.

symbol.type Specifies the type of this symbol table entry. Some possible types are *root*, *module*, *value*, *procedure*, and *widget*.

symbol.data Specifies data for the symbol table entry. The data varies with the type of the entry. Often it contains pointers to other symbol table entries, or the actual data for the data type.

prev.source.position
Specifies the end point in the source code for the previous source item.

source.position
Specifies the range of positions in the source code for this symbol.

The exact data structures for each symbol type are defined in the include file **UilSymDef.h**. Note that this file is automatically included when an application includes the file **UilDef.h**.

Related Information

Uil(3)

Chapter 9

File Formats

mwmrc

Purpose the mwm Window Manager Resource Description File

Description

The **mwmrc** window manager is a supplementary resource file that controls much of the behavior of the CDE window manager **mwm**. It contains descriptions of resources that cannot easily be written using standard X Window System, Version 11 resource syntax. The resource description file contains entries that are referred to by X resources in defaults files (for example, **/usr/mwm/app-defaults/$LANG/mwm**) or in the *RESOURCE_MANAGER* property on the root window. For example, the resource description file enables you to specify different types of window menus; however, an X resource is used to specify which of these window menus the **mwm** should use for a particular window. The specifications of the resource description file supported by the mwm workspace manager are a strict superset of the specifications supported by the OSF Motif Window Manager (**mwm 1.2.4**). In other words, the **system.mwmrc** or **$HOME/.mwmrc** file that you've used for **mwm** is easily made usable by **mwm**.

Location

The workspace manager searches for one of the following resource description files, where *$LANG* is the value of the language environment on a per-user basis:

```
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
/usr/lib/X11/$LANG/system.mwmrc
/usr/lib/X11/system.mwmrc
```

The first file found is the first used. If no file is found, a set of built-in specifications is used. A particular resource description file can be selected using the *configFile* resource. The following shows how a different resource description file can be specified from the command line:

```
/usr/mwm/bin/mwm -xrm "mwm*configFile: mymwmrc"
```

Resource Types

The following types of resources can be described in the mwm resource description file:

Buttons Workspace manager functions can be bound (associated) with button events.

Keys Workspace manager functions can be bound (associated) with key press events.

Menus Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

MWM Resource Description File Syntax

The **mwm** resource description file is a standard text file that contains items of information separated by blanks, tabs, and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (for example, the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes (" "). Single characters can be quoted by preceding them by the back-slash character (\), except for workspace names, which may contain no back-slash characters. If a line ends with a back-slash, the next line is considered a continuation of that line. All text from an unquoted **#** to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If **!** is the first character in a line, the line is regarded as a comment.

Workspace Manager Functions

Workspace manager functions can be accessed with button and key bindings, and with workspace manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

function = function_name [function_args]
function_name = workspace manager function
function_args = {quoted_item | unquoted_item}

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by **mwm** as **f.nop**.

f.beep This function causes a beep.

f.circle_down [*icon* | *window*]

This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it is no longer obscuring any other window or icon). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an *icon* function argument is specified, then the function applies only to icons. If a *window* function argument is specified then the function applies only to windows.

f.circle_up [*icon* | *window*]

This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.exec command (or *!* **command**)

This function causes *command* to be executed (using the value of the *$MWMSHELL* or *$SHELL* environment variable if set; otherwise, **/usr/bin/sh**). The *!* notation can be used in place of the **f.exec** function name.

f.focus_color

This function sets the colormap focus to a client window. If this function is done in a root context, then the default colormap (setup by the X Window System for the screen where **mwm** is running) is installed and there is no specific client window colormap focus. This function is treated as **f.nop** if *colormapFocusPolicy* is not explicit.

f.focus_key This function sets the keyboard input focus to a client window or icon. This function is treated as **f.nop** if *keyboardFocusPolicy* is not explicit or the function is executed in a root context.

f.kill This function is used to close application windows. The actual processing that occurs depends on the protocols that the application observes. The application lists the protocols it observes in the WM_PROTOCOLS property on its top level window. If the application observes the WM_DELETE_WINDOW protocol, it is sent a message that requests the window be deleted. If the application observes both WM_DELETE_WINDOW and WM_SAVE_YOURSELF, it is sent one message requesting the window be deleted and another message advising it to save its state. If the application observes only the WM_SAVE_YOURSELFprotocol , it is sent a message advising it to save its state. After a delay (specified by the resource *quitTimeout*), the application's connection to the X server is terminated. If the application observes neither of these protocols, its connection to the X server is terminated.

f.lower [– **client** | *within* | *freeFamily*]
This function lowers a primary window to the bottom of the global window stack (where it obscures no other window) and lowers the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive. The *client* argument indicates the name or class of a client to lower. The name or class of a client appears in the WM_CLASS property on the client's top-level window. If the *client* argument is not specified, the context that the function was invoked in indicates the window or icon to lower. Specifying *within* lowers the secondary window within the family (staying above the parent) but does not lower the client family in the global window stack. Specifying *freeFamily* lowers the window to the bottom of the global windows stack from its local family stack.

f.maximize This function causes a client window to be displayed with its maximum size. Refer to the *maximumClientSize*, *maximumMaximumSize*, and *limitResize* resources in **mwm**(1).

f.menu *menu_name*
This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The *menu_name* function argument identifies the menu to be used.

f.minimize This function causes a client window to be minimized (iconified). When a window is minimized with no icon box in use, and if the *lowerOnIconify* resource has the value True (the default), the icon is placed on the bottom of the window stack (such that it obscures no other

window). If an icon box is used, then the client's icon changes to its iconified form inside the icon box. Secondary windows (that is, transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move This function initiates an interactive move of a client window.

f.next_cmap This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [*icon* | *window* | *transient*]
This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the workspace manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if **keyboardFocusPolicy** is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, then transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is specified, then the function applies only to icons. If a *window* function argument is specified, then the function applies only to windows.

f.nop This function does nothing.

f.normalize This function causes a client window to be displayed with its normal size. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.normalize_and_raise
This function causes a client window to be displayed with its normal size and raised to the top of the window stack. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons
This function is used to relayout icons (based on the layout policy being used) on the root window or in the icon box. In general this causes icons to be "packed" into the icon grid.

f.pass_keys This function is used to enable/disable (toggle) processing of key bindings for workspace manager functions. When it disables key binding processing all keys are passed on to the window with the keyboard input focus and no workspace manager functions are invoked. If the

f.pass_keys function is invoked with a key binding to disable key binding processing the same key binding can be used to enable key binding processing.

f.post_wmenu

This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the window menu is placed at the top-left corner of the client window.

f.prev_cmap

This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key [*icon* | *window* | *transient*]

This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the workspace manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if *keyboardFocusPolicy* is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, then transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.quit_mwm

This function terminates mwm (but NOT the X window system).

f.raise [–*client* | *within* | *freeFamily*]

This function raises a primary window to the top of the global window stack (where it is obscured by no other window) and raises the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive. The *client* argument indicates the name or class of a client to lower. If the *client* is not specified, the context that the function was invoked in indicates the window or icon to lower. Specifying *within* raises the secondary window within the family but does not raise the client family in the global window stack. Specifying *freeFamily* raises the window to the top of its

local family stack and raises the family to the top of the global window stack.

f.raise_lower [*within* | *freeFamily*]
This function raises a primary window to the top of the global window stack if it is partially obscured by another window; otherwise, it lowers the window to the bottom of the window stack. The arguments to this function are mutually exclusive. Specifying *within* raises a secondary window within the family (staying above the parent window), if it is partially obscured by another window in the application's family; otherwise, it lowers the window to the bottom of the family stack. It has no effect on the global window stacking order. Specifying *freeFamily* raises the window to the top of its local family stack, if obscured by another window, and raises the family to the top of the global window stack; otherwise, it lowers the window to the bottom of its local family stack and lowers the family to the bottom of the global window stack.

f.refresh
This function causes all windows to be redrawn.

f.refresh_win
This function causes a client window to be redrawn.

f.resize This function initiates an interactive resize of a client window.

f.restore
This function restores the previous state of an icon's associated window. If a maximized window is iconified, then **f.restore** restores it to its maximized state. If a normal window is iconified, then **f.restore** restores it to its normalized state.

f.restore_and_raise
This function restores the previous state of an icon's associated window and raises the window to the top of the window stack. If a maximized window is iconified, then **f.restore_and_raise** restores it to its maximized state and raises it to the top of the window stack. If a normal window is iconified, then **f.restore_and_raise** restores it to its normalized state and raises it to the top of the window stack.

2f.restart This function causes mwm to be restarted (effectively terminated and re-executed). Restart is necessary for **mwm** to incorporate changes in both the **mwmrc** file and X resources.

f.screen [*next* | *prev* | *back* | *screen_number*]

This function causes the pointer to be warp to a specific screen number or to the *next*, *previous*, or last visited (*back*) screen. The arguments to this function are mutually exclusive. The *screen_number* argument indicates the screen number that the pointer is to be warped. Screens are numbered starting from screen 0. Specifying *next* cause the pointer to warp to the next managed screen (skipping over any unmanaged screens). Specifying *prev* cause the pointer to warp to the previous managed screen (skipping over any unmanaged screens). Specifying *back* cause the pointer to warp to the last visited screen.

f.send_msg *message_number*

This function sends an **XClientMessageEvent** of type _MOTIF_WM_MESSAGES with *message_type* set to *message_number*. The client message is sent only if *message_number* is included in the client's _MOTIF_WM_MESSAGES property. A menu item label is grayed out if the menu item is used to do **f.send_msg** of a message that is not included in the client's _MOTIF_WM_MESSAGES property.

f.separator This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior

This function causes the workspace manager to restart with the default behavior (if a custom behavior is configured) or a custom behavior (if a default behavior is configured). By default this is bound to *Shift Ctrl Alt <Key>!*.

f.title This function inserts a title in the menu pane at the specified location.

f.version This function causes the workspace manager to display its release version in a dialog box.

Function Constraints

Each function may be constrained as to which resource types can specify the function (for example, menu pane) and also what context the function can be used in (for example, the function is done to the selected client window). Function contexts are:

root No client window or icon has been selected as an object for the function.

window A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are

applied only when the window is in its normalized state (for example, **f.maximize**) or its maximized state (for example, **f.normalize**).

icon An icon has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as **f.nop**. The following table indicates the resource types and function contexts in which workspace manager functions apply.

Function	Contexts	Resources
f.beep	root,icon,window	button,key,menu
f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	root,icon,window	button,key,menu
f.kill	icon,window	button,key,menu
f.lower	root,icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.normalize_and_raise	icon,window	button,key,menu
f.pack_icons	root,icon,window	button,key,menu
f.pass_keys	root,icon,window	button,key,menu
f.post_wmenu	root,icon,window	button,key
f.prev_cmap	root,icon,window	button,key,menu
f.prev_key	root,icon,window	button,key,menu

f.quit_mwm	root	button,key,menu (root only)
f.raise	root,icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu
f.resize	window	button,key,menu
f.restart	root	button,key,menu (root only)
f.restore	icon,window	button,key,menu
f.restore_and_raise	icon,window	button,key,menu
f.screen	root,icon,window	button,key,menu
f.send_msg	icon,window	button,key,menu
f.separator	root,icon,window	menu
f.set_behavior	root,icon,window	button,key,menu
f.title	root,icon,window	menu
f.version	root,icon,window	button,key,menu

Workspace Manager Event Specification

Events are indicated as part of the specifications for button and key binding sets, and menu panes. Button events have the following syntax:

button =~[modifier_list]<button_event_name >
modifier_list =~modifier_name { modifier_name}

The following table indicates the values that can be used for **modifier_name**. Note that [Alt] and [Meta] can be used interchangably on some hardware.

Modifier	Description
Ctrl	Control Key
Shift	Shift Key
Alt	Alt Key
Meta	Meta Key
Mod1	Modifier1

Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

Locking modifiers are ignored when processing button and key bindings. The following table lists keys that are interpreted as locking modifiers. The X server may map some of these symbols to the Mod1 - Mod5 modifier keys. These keys may or may not be available on your hardware: Key Symbol Caps Lock Shift Lock Kana Lock Num Lock Scroll Lock The following table indicates the values that can be used for **button_event_name**.

Button	Description
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release
Btn2Click2	Button 2 Double Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double Click
Btn5Down	Button 5 Press

Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double Click

Key events that are used by the workspace manager for menu mnemonics and for binding to workspace manager functions are single key presses; key releases are ignored. Key events have the following syntax:

key =~[**modifier_list**] Key**key_name**
modifier_list =~**modifier_name** { **modifier_name**}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The **key_name** is an X11 keysym name. Keysym names can be found in the **keysymdef.h** file (remove the *XK_* prefix).

Button Bindings

The **buttonBindings** resource value is the name of a set of button bindings that are used to configure workspace manager behavior. A workspace manager function can be done when a button press occurs with the pointer over a framed client window, an icon or the root window. The context for indicating where the button press applies is also the context for invoking the workspace manager function when the button press is done (significant for functions that are context sensitive). The button binding syntax is

```
Buttons bindings_set_name
{
    button      context     function
    button      context     function
    ...
    button      context     function
}
```

The syntax for the **context** specification is: **context** = **object**[| **context**] **object** = *root* | *icon* | *window* | *title* | *frame* | *border* | *app* The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of *window* indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The *frame* context is for the window management frame around a client window (including the border and titlebar), the *border* context

is for the border part of the window management frame (not including the titlebar), the *title* context is for the title area of the window management frame, and the *app* context is for the application window (not including the window management frame). If an **f.nop** function is specified for a button binding, the button binding is not done.

Key Bindings

The **keyBindings** resource value is the name of a set of key bindings that are used to configure workspace manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings. The key binding syntax is:

```
Keys bindings_set_name
{
    key     context     function
    key     context     function
    . . .
    key     context     function
}
```

If an **f.nop** function is specified for a key binding, the key binding is not done. If an **f.post_wmenu** or **f.menu** function is bound to a key, **mwm** automatically uses the same key for removing the menu from the screen after it has been popped up. The **context** specification syntax is the same as for button bindings with one addition. The context *ifkey* may be specified for binding keys that may not be available on all displays. If the key is not available and if *ifkey* is in the context, then reporting of the error message to the error log is suppressed. This feature is useful for networked, heterogeneous environments. For key bindings, the *frame*, *title*, *border*, and *app* contexts are equivalent to the *window* context. The context for a key event is the window or icon that has the keyboard input focus (*root* if no window or icon has the keyboard input focus).

Menu Panes

Menus can be popped up using the **f.post_wmenu** and **f.menu** workspace manager functions. The context for workspace manager functions that are done from a menu is *root*, *icon* or *window* depending on how the menu was popped up. In the case of the *window* menu or menus popped up with a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu. The menu pane specification syntax is:

```
Menu menu_name
{
    label   [mnemonic]   [accelerator ]    function
    label   [mnemonic]   [accelerator ]    function
    ...
    label   [mnemonic]   [accelerator ]    function
}
```

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies. The **label** may be a string or a bitmap file. The label specification has the following syntax:

```
label = text | bitmap_file
bitmap_file = @file_name
text = quoted_item | unquoted_item
```

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the **f.nop** function or an invalid function or a function that doesn't apply in the current context. A **mnemonic** specification has the following syntax:

```
mnemonic = _ character
```

The first matching **character** in the label is underlined. If there is no matching **character** in the label, no mnemonic is registered with the workspace manager for that label. Although the **character** must exactly match a character in the label, the mnemonic does not execute if any modifier (such as Shift) is pressed with the character key. The **accelerator** specification is a key event specification with the same syntax as is used for key bindings to workspace manager functions.

Including Files

You may include other files into your mwmrc file by using the *include* construct. For example,

```
INCLUDE
{
    /usr/local/shared/mwm.menus
    /home/kmt/personal/my.bindings}
```

causes the files named to be read in and interpreted in order as an additional part of the mwmrc file. *Include* is a top-level construct. It cannot be nested inside another construct.

WARNINGS

Errors that occur during the processing of the resource description file are recorded in: **$HOME/.mwm/errorlog**. Be sure to check this file if the appearance or behavior of **mwm** is not what you expect.

Files

$HOME/$LANG/.mwmrc $HOME/.mwmrc /usr/lib/X11/$LANG/system.mwmrc /usr/lib/X11/system.mwmrc $HOME/.mwm/errorlog

Related Information

mwm(1), **mwm(**1X), **X**(1).

Traits

Purpose

Lists the traits used by the Motif Toolkit.

Description

A trait is a characteristic of a widget. A widget holding a particular trait is announcing a particular ability to other widgets. The following table summarizes the standard Motif traits.

Purpose of Each Trait	
Trait Name	**A Widget Holding This Trait Can Do The Following:**
XmQTaccessTextual	Display one primary text parcel.
XmQTactivatable	Become a command button in a dialog box.
XmQTcareParentVisual	Borrow its parent's visual information.
XmQTcontainer	Manage container item children.
XmQTcontainerItem	Become a child of a container widget.
XmQTdialogShellSavvy	Become a child of a DialogShell.
XmQTjoinSide	Attach itself to one side of a suitable parent.
XmQTmenuSavvy	Become a menu child.
XmQTmenuSystem	Manage a menu system.
XmQTnavigator	Act as a navigator to a scrollable widget.
XmQTscrollFrame	Handle one or more navigator widgets.
XmQTspecifyRenderTable	Supply the names of its default render tables.

XmQTtakesDefault	Change its appearance to show that it is the default button.
XmQTtransfer	Transfer data to other widgets and/or receive data from other widgets

Traits are not often used in Motif application programs. However, traits are very important to widget writers. For complete details on traits, see the *Motif 2.1—Widget Writer's Guide*.

The following table lists the names of all widgets and gadgets in the standard Motif widget set that hold a particular trait. For example, the following table shows that the *XmQTcontainerItem* trait is held by the **XmIconGadget**. As the table suggests, some traits are held by many of the standard Motif widgets.

Trait Installation in Standard Widget Set	
Trait Name	**Is Installed on The Following Widgets:**
XmQTaccessTextual	*XmLabel* and all its subclasses; **XmLabelGadget** and all its subclasses; *XmText; XmTextField*
XmQTactivatable	*XmArrowButton; XmArrowButtonGadget; XmDrawnButton; XmPushButton; XmPushButtonGadget*
XmQTcareParentVisual	All the subclasses of **XmGadget** (but not **XmGadget** itself); **XmPrimitive** and all its subclasses
XmQTcontainer	**XmContainer**
XmQTcontainerItem	**XmIconGadget**
XmQTdialogShellSavvy	**XmBulletinBoard**
XmQTjoinSide	No widgets install this trait

XmQTmenuSavvy	*XmLabel; XmDrawnButton; XmCascadeButton; XmPushButton; XmToggleButton; XmLabelGadget; XmCascadeButtonGadget; XmPushButtonGadget; XmToggleButtonGadget*
XmQTmenuSystem	**XmRowColumn**
XmQTnavigator	*XmScrollBar; XmSpinBox*
XmQTscrollFrame	*XmNotebook; XmScrolledWindow*
XmQTspecifyRenderTable	**XmBulletinBoard** and all its subclasses; *XmMenuShell; XmVendorShell*
XmQTtakesDefault	*XmPushButton; XmPushButtonGadget*
XmQTtransfer	*XmContainer; XmLabel* and all its subclasses; **XmLabelGadget** and all its subclasses; *XmList; XmScale; XmText; XmTextField*

The following table lists the traits installed on each widget. For example, the following table indicates that the **XmArrowButton** widget holds both the *XmQTactivatable* and *XmQTcareParentVisual* traits.

Trait Use by Widget	
Widget Name	**Installs These Traits**
XmArrowButton	*XmQTactivatable, XmQTcareParentVisual*
XmArrowButtonGadget	*XmQTactivatable, XmQTcareParentVisual*
XmBulletinBoard	*XmQTdialogShellSavvy, XmQTspecifyRenderTable*
XmCascadeButton	*XmQTaccessTextual, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtransfer*

XmCascadeButtonGadget	*XmQTaccessTextual, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtransfer XmComboBox*
XmCommand	*XmQTspecifyRenderTable*
XmContainer	*XmQTcontainer, XmQTtransfer*
XmDialogShell	None
XmDisplay	None
XmDragContext	None
XmDragIcon	None
XmDrawingArea	None
XmDrawnButton	*XmQTaccessTextual, XmQTactivatable, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtransfer*
XmDropTransfer	None
XmFileSelectionBox	*XmQTspecifyRenderTable*
XmForm	*XmQTspecifyRenderTable*
XmFrame	None
XmGadget	None
XmIconGadget	*XmQTcareParentVisual, XmQTcontainerItem*
XmLabel	*XmQTaccessTextual, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtransfer*
XmLabelGadget	*XmQTaccessTextual, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtransfer*
XmList	*XmQTcareParentVisual, XmQTtransfer*
XmMainWindow	
XmManager	

XmMenuShell	*XmQTspecifyRenderTable*
XmMessageBox	*XmQTspecifyRenderTable*
XmNotebook	*XmQTscrollFrame*
XmPanedWindow	
XmPrimitive	*XmQTcareParentVisual*
XmPushButton	*XmQTaccessTextual, XmQTactivatable, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtakesDefault, XmQTtransfer*
XmPushButtonGadget	*XmQTaccessTextual, XmQTactivatable, XmQTcareParentVisual, XmQTmenuSavvy, XmQTtakesDefault, XmQTtransfer*
XmRowColumn	*XmQTmenuSystem*
XmScale	*XmQTtransfer*
XmScreen	
XmScrollbar	*XmQTcareParentVisual, XmQTnavigator*
XmScrolledWindow	*XmQTscrollFrame*
XmSelectionBox	*XmQTspecifyRenderTable*
XmSeparator	*XmQTcareParentVisual*
XmSeparatorGadget	*XmQTcareParentVisual*
XmSpinBox	*XmQTnavigator*
XmText	*XmQTaccessTextual, XmQTcareParentVisual, XmQTtransfer*
XmTextField	*XmQTaccessTextual, XmQTcareParentVisual, XmQTtransfer*
XmToggleButton	*XmQTaccessTextual, XmQTcareParentVisual, XmQTtransfer*

XmToggleButtonGadget	*XmQTaccessTextual, XmQTcareParentVisual, XmQTtransfer*
VendorShell	*XmQTspecifyRenderTable*

The following table summarizes how the standard Motif widgets access traits. There are two general ways for a widget to access the traits of another widget.

One way is for a widget to ask another widget if it holds a particular trait. For example, **XmBulletinBoard** asks each of its children widgets if they hold the *XmQTtakesDefault* trait. **XmBulletinBoard** calls none of the trait methods of *XmQTtakesDefault*.

Another kind of access is when one widget calls another widget's trait method(s). For example, **XmBulletinBoard** calls the **getRenderTable** trait method of the *XmQTspecifyRenderTable* trait.

Trait Access By Widget		
Widget	**Accesses These Traits:**	**Calls These Trait Methods:**
XmArrowButton	None	None
XmArrowButtonGadget	None	None
XmBulletinBoard	*XmQTtakesDefault*	None
XmBulletinBoard	*XmQTspecifyRenderTable*	**getRenderTable**
XmCascadeButton	*XmQTmenuSystem*	Many
XmCascadeButton	*XmQTspecifyRenderTable*	**getRenderTable**
XmCascadeButtonGadget	*XmQTmenuSystem*	Many
XmCascadeButtonGadget	*XmQTspecifyRenderTable*	**getRenderTable**
XmComboBox	*XmQTaccessTextual*	**getValue, setValue**
XmCommand	None	None
XmContainer	*XmQTcontainerItem*	**getValues, setValues**
XmContainer	*XmQTscrollFrame*	**getInfo**
XmDialogShell	*XmQTdialogShellSavvy*	**callMapUnmapCB**
XmDisplay	None	None
XmDragContext	None	None
XmDragIcon	None	None
XmDrawingArea	None	None

XmDrawnButton	*XmQTmenuSystem*	Many
XmDrawnButton	*XmQTspecifyRenderTable*	**getRenderTable**
XmDropTransfer	None	None
XmFileSelectionBox	*XmQTactivatable*	None
XmForm	None	None
XmFrame	None	None
XmGadget	None	None
XmIconGadget	**XmQTcontainer**	**getValues**
XmIconGadget	*XmQTspecifyRenderTable*	**getRenderTable**
XmLabel	*XmQTmenuSystem*	**various methods**
XmLabel	*XmQTspecifyRenderTable*	**getRenderTable**
XmLabelGadget	*XmQTmenuSystem*	**various methods**
XmLabelGadget	*XmQTspecifyRenderTable*	**getRenderTable**
XmList	*XmQTnavigator*	**getValues**
XmList	*XmQTscrollFrame*	**getInfo, init**
XmList	*XmQTspecifyRenderTable*	**getRenderTable**
XmMainWindow	*XmQTmenuSystem*	**various methods**
XmManager	None	None
XmMenuShell	*XmQTmenuSystem*	**various methods**
XmMenuShell	*XmQTspecifyRenderTable*	**getRenderTable**
XmMessageBox	*XmQTactivatable*	None
XmNotebook	*XmQTscrollFrame*	**init, addNavigator, removeNavigator**
XmNotebook	*XmQTnavigator*	**getValue**
XmNotebook	*XmQTactivatable*	**changeCB**
XmNotebook	*XmQTaccessTextual*	None
XmPanedWindow	None	None
XmPrimitive	None	None
XmPushButton	*XmQTmenuSystem*	**various methods**
XmPushButton	*XmQTspecifyRenderTable*	**getRenderTable**

XmPushButtonGadget	*XmQTmenuSystem*	**various methods**
XmPushButtonGadget	*XmQTspecifyRenderTable*	**getRenderTable**
XmRowColumn	*XmQTmenuSavvy*	**getAccelerator, getMnemonic, getActivateCBName**
XmRowColumn	*XmQTmenuSystem*	**various methods**
XmScale	*XmQTspecifyRenderTable*	**getRenderTable**
XmScreen	None	None
XmScrollbar	None	None
XmScrolledWindow	*XmQTnavigator*	**getValue**
XmScrolledWindow	*XmQTscrollFrame*	**init, addNavigator**
XmSelectionBox	*XmQTaccessTextual*	**setValue;**
XmSelectionBox	*XmQTactivatable*	None
XmSeparator	None	None
XmSeparatorGadget	None	None
XmSpinBox	*XmQTaccessTextual*	**setValue**
XmText	*XmQTaccessTextual*	**getValue, setValue**
XmText	*XmQTnavigator*	**getValue**
XmText	*XmQTscrollFrame*	**getInfo, init**
XmText	*XmQTspecifyRenderTable*	**getRenderTable**
XmTextField	*XmQTspecifyRenderTable*	**getRenderTable**
XmToggleButton	*XmQTmenuSystem*	**various methods**
XmToggleButton	*XmQTspecifyRenderTable*	**getRenderTable**
XmToggleButtonGadget	*XmQTmenuSystem*	**various methods**
XmToggleButtonGadget	*XmQTspecifyRenderTable*	**getRenderTable**
VendorShell	*XmQTspecifyRenderTable*	**getRenderTable**

Related Information

The following reference pages are documented in the *Motif 2.1—Widget Writer's Guide*: **XmeTraitSet**(3), **XmeTraitGet**(3), **XmQTaccessTextual**(3), **XmQTactivatable**(3), **XmQTcareParentVisual**(3), **XmQTcontainer**(3),

XmQTcontainerItem(3), **XmQTdialogShellSavvy**(3), **XmQTjoinSide**(3), **XmQTmenuSavvy**(3), **XmQTmenuSystem**(3), **XmQTnavigator**(3), **XmQTscrollFrame**(3), **XmQTspecifyRenderTable**(3), and **XmQTtakesDefault**(3).

UIL

Purpose

The user interface language file format

Synopsis

MODULE *module_name*
[NAMES = CASE_INSENSITIVE | CASE_SENSITIVE]
[CHARACTER_SET = character_set]
[OBJECTS = { *widget_name* = GADGET | WIDGET; [...] }]
{ [
[*value_section*] |
[*procedure_section*] |
[*list_section*] |
[*object_section*] |
[*identifier_section*]
[...]
] }
END MODULE;

Description

The UIL language is used for describing the initial state of a user interface for a widget based application. UIL describes the widgets used in the interface, the resources of those widgets, and the callbacks of those widgets. The UIL file is compiled into a UID file using the command **uil** or by the callable compiler **Uil()**. The contents of the compiled UID file can then be accessed by the various Motif Resource Management (MRM) functions from within an application program.

The UID file is independent of the platform on which the Motif program will eventually be run. In other words, the same UID file can be used on any system that can run Motif.

File

A UIL file consists of a single complete module, described in the syntax description above, or, if the file is to be included in a larger UIL file, one complete "section,"

as described below. UIL uses five different kinds of sections: value, procedure, list, object, and identifier.

UIL is a free-form language. This means that high-level constructs such as object and value declarations do not need to begin in any particular column and can span any number of lines. Low-level constructs such as keywords and punctuation characters can also begin in any column; however, except for string literals and comments, they cannot span lines.

The UIL compiler accepts input lines up to 132 characters in length.

MODULE module_name

The name by which the UIL module is known in the UID file. This name is stored in the UID file for later use in the retrieval of resources by the MRM. This name is always stored in uppercase in the UID file.

NAMES = *CASE_INSENSITIVE* | *CASE_SENSITIVE*

Indicates whether names should be treated as case sensitive or case insensitive. The default is case sensitive. The case-sensitivity clause should be the first clause in the module header, and in any case must precede any statement that contains a name. If names are case sensitive in a UIL module, UIL keywords in that module must be in lowercase. Each name is stored in the UIL file in the same case as it appears in the UIL module. If names are case insensitive, then keywords can be in uppercase, lowercase, or mixed case, and the uppercase equivalent of each name is stored in the UID file.

CHARACTER_SET = **character_set**

Specifies the default character set for string literals in the module that do not explicitly set their character set. The default character set, in the absence of this clause is the codeset component of the *LANG* environment variable, or the value of **XmFALLBACK_CHARSET** if *LANG* is not set or has no codeset component. The value of **XmFALLBACK_CHARSET** is defined by the UIL supplier, but is usually ISO8859-1 (equivalent to ISO_LATIN1). Use of this clause turns off all localized string literal processing turned on by the compiler flag **-s** or the **Uil_command_type** data structure element **use_setlocale_flag**.

OBJECTS = { *widget_name* = *GADGET* | **WIDGET; }**

Indicates whether the widget or gadget form of the control specified by *widget_name* is used by default. By default the widget form is used, so the gadget keyword is usually the only one used. The specified control should be one that has both a widget and gadget

version: XmCascadeButton, XmLabel, XmPushButton, XmSeparator, and XmToggleButton. The form of more than one control can be specified by delimiting them with semicolons. The gadget or widget form of an instance of a control can be specified with the *GADGET* and *WIDGET* keywords in a particular object declaration.

value_section
: Provides a way to name a value expression or literal. The value name can then be referred to by declarations that occur elsewhere in the UIL module in any context where a value can be used. Values can be forward referenced. Value sections are described in more detail later in the reference page.

procedure_section
: Defines the callback routines used by a widget and the creation routines for user-defined widgets. These definitions are used for error checking. Procedure sections are described in more detail later in the reference page.

list_section
: Provides a way to group together a set of arguments, controls (children), callbacks, or procedures for later use in the UIL module. Lists can contain other lists, so that you can set up a hierarchy to clearly show which arguments, controls, callbacks, and procedures are common to which widgets. List sections are described in more detail later in the reference page.

object_section
: Defines the objects that make up the user interface of the application. You can reference the object names in declarations that occur elsewhere in the UIL module in any context where an object name can be used (for example, in a controls list, as a symbolic reference to a widget ID, or as the *tag_value* argument for a callback procedure). Objects can be forward referenced. Object sections are described in more detail later in the reference page.

identifier_section
: Defines a run-time binding of data to names that appear in the UIL module. Identifier sections are described in more detail later in the reference page.

The UIL file can also contain comments and include directives, which are described along with the main elements of the UIL file format in the following sections.

Comments

Comments can take one of two forms, as follows:

- The comment is introduced with the sequence **/*** followed by the text of the comment and terminated with the sequence ***/**. This form of comment can span multiple source lines.
- The comment is introduced with an ! (exclamation point), followed by the text of the comment and terminated by the end of the source line.

Neither form of comment can be nested.

Value sections

A value section consists of the keyword *VALUE* followed by a sequence of value declarations. It has the following syntax:

VALUE value_name **:** [*EXPORTED* | *PRIVATE*]*value_expression* | *IMPORTED value_type* **;**

Where *value_expression* is assigned to *value_name* or a *value_type* is assigned to an imported value name. A value declaration provides a way to name a value expression or literal. The value name can be referred to by declarations that occur later in the UIL module in any context where a value can be used. Values can be forward referenced.

EXPORTED A value that you define as exported is stored in the UID file as a named resource, and therefore can be referenced by name in other UID files. When you define a value as exported, MRM looks outside the module in which the exported value is declared to get its value at run time.

PRIVATE A private value is a value that is not imported or exported. A value that you define as private is not stored as a distinct resource in the UID file. You can reference a private value only in the UIL module containing the value declaration. The value or object is directly incorporated into anything in the UIL module that references the declaration.

IMPORTED A value that you define as imported is one that is defined as a named resource in a UID file. MRM resolves this declaration with the corresponding exported declaration at application run time.

By default, values and objects are private. The following is a list of the supported value types in UIL:

- *ANY*
- *ARGUMENT*

- *BOOLEAN*
- *COLOR*
- *COLOR_TABLE*
- *COMPOUND_STRING*
- *FLOAT*
- *FONT*
- *FONT_TABLE*
- *FONTSET*
- *ICON*
- *INTEGER*
- *INTEGER_TABLE*
- *KEYSYM*
- *REASON*
- *SINGLE_FLOAT*
- *STRING*
- *STRING_TABLE*
- *TRANSLATION_TABLE*
- *WIDE_CHARACTER*
- *WIDGET*

Procedure sections

A procedure section consists of the keyword *PROCEDURE* followed by a sequence of procedure declarations. It has the following syntax:

PROCEDURE
procedure_name [([*value_type*])];

Use a procedure declaration to declare

- A routine that can be used as a callback routine for a widget
- The creation function for a user-defined widget

You can reference a procedure name in declarations that occur later in the UIL module in any context where a procedure can be used. Procedures can be forward referenced. You cannot use a name you used in another context as a procedure name.

In a procedure declaration, you have the option of specifying that a parameter will be passed to the corresponding callback routine at run time. This parameter is called the callback tag. You can specify the data type of the callback tag by putting the data type in parentheses following the procedure name. When you compile the module, the UIL compiler checks that the argument you specify in references to the procedure is of this type. Note that the data type of the callback tag must be one of the valid UIL data types. You can use a widget as a callback tag, as long as the widget is defined in the same widget hierarchy as the callback, that is they have a common ancestor that is in the same UIL hierarchy.

The following list summarizes how the UIL compiler checks argument type and argument count, depending on the procedure declaration.

No parameters
: No argument type or argument count checking occurs. You can supply either 0 or one arguments in the procedure reference.

( )
: Checks that the argument count is 0 (zero).

(ANY)
: Checks that the argument count is 1. Does not check the argument type. Use the *ANY* type to prevent type checking on procedure tags.

(*type***)**
: Checks for one argument of the specified type.

(class_name)
: Checks for one widget argument of the specified widget class.

While it is possible to use any UIL data type to specify the type of a tag in a procedure declaration, you must be able to represent that data type in the programming language you are using. Some data types (such as integer, Boolean, and string) are common data types recognized by most programming languages. Other UIL data types (such as string tables) are more complicated and may require that you set up an appropriate corresponding data structure in the application in order to pass a tag of that type to a callback routine.

You can also use a procedure declaration to specify the creation function for a user-defined widget. In this case, you specify no formal parameters. The procedure is invoked with the standard three arguments passed to all widget creation functions. (See the Motif Toolkit documentation for more information about widget creation functions.)

List sections

A list section consists of the keyword *LIST* followed by a sequence of list declarations. It has the following syntax:

```
LIST
    list_name: { list_item; [...] }
    [...]
```

You can also use list sections to group together a set of arguments, controls (children), callbacks, or procedures for later use in the UIL module. Lists can contain other lists, so that you can set up a hierarchy to clearly show which arguments, controls, callbacks, and procedures are common to which widgets. You cannot mix the different types of lists; a list of a particular type cannot contain entries of a different list type or reference the name of a different list type. A list name is always private to the UIL module in which you declare the list and cannot be stored as a named resource in a UID file.

The additional list types are described in the following sections.

Arguments List Structure

An arguments list defines which arguments are to be specified in the arguments list parameter when the creation routine for a particular object is called at run time. An arguments list also specifies the values for those arguments. Argument lists have the following syntax:

```
LIST
    list_name: ARGUMENTS {
        argument_name = value_expression;
        [...] }
[...]
```

The argument name must be either a built-in argument name or a user-defined argument name that is specified with the *ARGUMENT* function.

If you use a built-in argument name as an arguments list entry in an object definition, the UIL compiler checks the argument name to be sure that it is supported by the type of object that you are defining. If the same argument name appears more than once in a given arguments list, the last entry that uses that argument name supersedes all previous entries with that name, and the compiler issues a message.

Some arguments, such as **XmNitems** and **XmNitemCount**, are coupled by the UIL compiler. When you specify one of the arguments, the compiler also sets the other. The coupled argument is not available to you.

The Motif Toolkit and the X Toolkit (intrinsics) support constraint arguments. A constraint argument is one that is passed to children of an object, beyond those arguments normally available. For example, the Form widget grants a set of constraint arguments to its children. These arguments control the position of the children within the Form.

Unlike the arguments used to define the attributes of a particular widget, constraint arguments are used exclusively to define additional attributes of the children of a particular widget. These attributes affect the behavior of the children within their parent. To supply constraint arguments to the children, you include the arguments in the arguments list for the child.

See **Appendix B** for information about which arguments are supported by which widgets. See **Appendix C** for information about what the valid value type is for each built-in argument.

Callbacks List Structure

Use a callbacks list to define which callback reasons are to be processed by a particular widget at run time. Callback lists have the following syntax:

LIST list_name **: CALLBACKS {** *reason_name* **= PROCEDURE** *procedure_name* **[(** [*value_expression*]**)];** | *reason_name* **=** *procedure_list* **;** [...] **}** [...]

For Motif Toolkit widgets, the reason name must be a built-in reason name. For a user-defined widget, you can use a reason name that you previously specified using the *REASON* function. If you use a built-in reason in an object definition, the UIL compiler ensures that reason is supported by the type of object you are defining. Appendix B shows which reasons each object supports.

If the same reason appears more than once in a callbacks list, the last entry referring to that name supersedes all previous entries using the same reason, and the UIL compiler issues a diagnostic message.

If you specify a named value for the procedure argument (callback tag), the data type of the value must match the type specified for the callback tag in the corresponding procedure declaration. When specifying a widget name as a procedure value expression you must also specify the type of the widget and a space before the name of the widget.

Because the UIL compiler produces a UID file rather than an object module (.o), the binding of the UIL name to the address of the entry point to the procedure is not done by the loader, but is established at run time with the MRM function **MrmRegisterNames**. You call this function before fetching any objects, giving it both the UIL names and the procedure addresses of each callback. The name you

register with MRM in the application program must match the name you specified for the procedure in the UIL module.

Each callback procedure receives three arguments. The first two arguments have the same form for each callback. The form of the third argument varies from object to object.

The first argument is the address of the data structure maintained by the Motif Toolkit for this object instance. This address is called the widget ID for this object.

The second argument is the address of the value you specified in the callbacks list for this procedure. If you do not specify an argument, the address is NULL. Note that, in the case where the value you specified is a string or an **XmString**, the value specified in the callbacks list already represents an address rather than an actual value. In the case of a simple string, for example, the value is the address of the first character of that string. In these cases, UIL does not add a level of indirection, and the second argument to the callback procedure is simply the value as specified in the callbacks list.

The third argument is the reason name you specified in the callbacks list.

Controls List Structure

A controls list defines which objects are children of, or controlled by, a particular object. Each entry in a controls list has the following syntax:

```
LIST
    list_name: CONTROLS {
        [child_name: ] [MANAGED | UNMANAGED] object_definition;
        [...] }
    [...]
```

If you specify the keyword *MANAGED* at run time, the object is created and managed; if you specify *UNMANAGED* at run time, the object is only created. Objects are managed by default.

You can use *child_name* to specify resources for the automatically created children of a particular control. Names for automatically created children are formed by appending **Xm_** to the name of the child widget. This name is specified in the documentation for the parent widget.

Unlike the arguments list and the callbacks list, a controls list entry that is identical to a previous entry does not supersede the previous entry. At run time, each controls list entry causes a child to be created when the parent is created. If the same object

definition is used for multiple children, multiple instances of the child are created at run time. See **Appendix B** for a list of which widget types can be controlled by which other widget types.

Procedures List Structure

You can specify multiple procedures for a callback reason in UIL by defining a procedures list. Just as with other list types, procedures lists can be defined in-line or in a list section and referenced by name.

If you define a reason more than once (for example, when the reason is defined both in a referenced procedures list and in the callbacks list for the object), previous definitions are overridden by the latest definition. The syntax for a procedures list is as follows:

```
LIST
    list_name: PROCEDURES {
        procedure_name [ ( [ value_expression ]) ];
        [...] }
    [...]
```

When specifying a widget name as a procedure value expression you must also specify the type of the widget and a space before the name of the widget.

Object Sections

An object section consists of the keyword *OBJECT* followed by a sequence of object declarations. It has the following syntax:

```
OBJECT object_name:
    [ EXPORTED | PRIVATE | IMPORTED ] object_type
        [ PROCEDURE creation_function ]
        [ object_name [ WIDGET | GADGET ] | {list_definitions } ]
```

Use an object declaration to define the objects that are to be stored in the UID file. You can reference the object name in declarations that occur elsewhere in the UIL module in any context where an object name can be used (for example, in a controls list, as a symbolic reference to a widget ID, or as the *tag_value* argument for a callback procedure). Objects can be forward referenced; that is, you can declare an object name after you reference it. All references to an object name must be consistent with the type of the object, as specified in the object declaration. You can specify an object as exported, imported, or private.

The object definition can contain a sequence of lists that define the arguments, hierarchy, and callbacks for the widget. You can specify only one list of each type for an object. When you declare a user-defined widget, you must include a reference to the widget creation function for the user-defined widget.

Note: Several widgets in the Motif Toolkit actually consist of two linked widgets. For example, *XmScrolledText* and *XmScrolledList* each consist of children **XmText** and **XmList** widgets under a **XmScrolledWindow** widget. When such a widget is created, its resources are available to both of the underlying widgets. This can occasionally cause problems, as when the programmer wants a **XmNdestroyCallback** routine named to act when the widget is destroyed. In this case, the callback resource will be available to both sub-widgets, and will cause an error when the widget is destroyed. To avoid these problems, the programmer should separately create the parent and child widgets, rather than relying on these linked widgets.

Use the *GADGET* or *WIDGET* keyword to specify the object type or to override the default variant for this object type. You can use the Motif Toolkit name of an object type that has a gadget variant (for example, **XmLabelGadget**) as an attribute of an object declaration. The *object_type* can be any object type, including gadgets. You need to specify the *GADGET* or *WIDGET* keyword only in the declaration of an object, not when you reference the object. You cannot specify the *GADGET* or *WIDGET* keyword for a user-defined object; user-defined objects are always widgets.

Identifier sections

The identifier section allows you to define an identifier, a mechanism that achieves run-time binding of data to names that appear in a UIL module. The identifier section consists of the reserved keyword *IDENTIFIER*, followed by a list of names, each name followed by a semicolon.

IDENTIFIER identifier_name; [...;]

You can later use these names in the UIL module as either the value of an argument to a widget or the tag value to a callback procedure. At run time, you use the MRM functions **MrmRegisterNames** and **MrmRegisterNamesInHierarchy** to bind the identifier name with the data (or, in the case of callbacks, with the address of the data) associated with the identifier.

Each UIL module has a single name space; therefore, you cannot use a name you used for a value, object, or procedure as an identifier name in the same module.

The UIL compiler does not do any type checking on the use of identifiers in a UIL module. Unlike a UIL value, an identifier does not have a UIL type associated with

it. Regardless of what particular type a widget argument or callback procedure tag is defined to be, you can use an identifier in that context instead of a value of the corresponding type.

To reference these identifier names in a UIL module, you use the name of the identifier wherever you want its value to be used.

Include directives

The include directive incorporates the contents of a specified file into a UIL module. This mechanism allows several UIL modules to share common definitions. The syntax for the include directive is as follows:

INCLUDE FILE *file_name*;

The UIL compiler replaces the include directive with the contents of the include file and processes it as if these contents had appeared in the current UIL source file.

You can nest include files; that is, an include file can contain include directives. The UIL compiler can process up to 100 references (including the file containing the UIL module). Therefore, you can include up to 99 files in a single UIL module, including nested files. Each time a file is opened counts as a reference, so including the same file twice counts as two references.

The *file_name* is a simple string containing a file specification that identifies the file to be included. The rules for finding the specified file are similar to the rules for finding header, or **.h** files using the include directive, **#include**, with a quoted string in C. The UIL uses the **--I** option for specifying a search directory for include files.

- If you do not supply a directory, the UIL compiler searches for the include file in the directory of the main source file.
- If the compiler does not find the include file there, the compiler looks in the same directory as the source file.
- If you supply a directory, the UIL compiler searches only that directory for the file.

Names and Strings

Names can consist of any of the characters A to Z, a to z, 0 to 9, $ (dollar sign), and _ (underscore). Names cannot begin with a digit (0 to 9). The maximum length of a name is 31 characters.

UIL gives you a choice of either case-sensitive or case-insensitive names through a clause in the *MODULE* header. For example, if names are case sensitive, the names

"sample" and "Sample" are distinct from each other. If names are case insensitive, these names are treated as the same name and can be used interchangeably. By default, UIL assumes names are case sensitive.

In **CASE-INSENSITIVE** mode, the compiler outputs all names in the UID file in uppercase form. In **CASE-SENSITIVE** mode, names appear in the UIL file exactly as they appear in the source.

The following table lists the reserved keywords, which are not available for defining programmer defined names.

Reserved Keywords			
ARGUMENTS	CALLBACKS	CONTROLS	END
EXPORTED	FALSE	GADGET	IDENTIFIER
INCLUDE	LIST	MODULE	OFF
ON	OBJECT	PRIVATE	PROCEDURE
PROCEDURES	TRUE	VALUE	WIDGET

The UIL unreserved keywords are described in the following list and table. These keywords can be used as programmer defined names, however, if you use any keyword as a name, you cannot use the UIL-supplied usage of that keyword.

- Built-in argument names (for example, **XmNx**, **XmNheight**)
- Built-in reason names (for example, **XmNactivateCallback**, **XmNhelpCallback**)
- Character set names (for example, *ISO_LATIN1*, *ISO_HEBREW_LR*)
- Constant value names (for example, **XmMENU_OPTION**, **XmBROWSE_SELECT**)
- Object types (for example, **XmPushButton**, **XmBulletinBoard**)

Unreserved Keywords		
ANY	ARGUMENT	ASCIZ_STRING_TABLE
ASCIZ_TABLE	BACKGROUND	BOOLEAN
CASE_INSENSITIVE	CASE_SENSITIVE	CHARACTER_SET
COLOR	COLOR_TABLE	COMPOUND_STRING

COMPOUND_STRING_-COMPONENT	COMPOUND_STRING_TABLE	FILE
FLOAT	FONT	FONT_TABLE
FONTSET	FOREGROUND	ICON
IMPORTED	INTEGER	INTEGER_TABLE
KEYSYM	MANAGED	NAMES
OBJECTS	REASON	RGB
RIGHT_TO_LEFT	SINGLE_FLOAT	STRING
STRING_TABLE	TRANSLATION_TABLE	UNMANAGED
USER_DEFINED	VERSION	WIDE_CHARACTER
WIDGET	XBITMAPFILE	

String literals can be composed of the uppercase and lowercase letters, digits, and punctuation characters. Spaces, tabs, and comments are special elements in the language. They are a means of delimiting other elements, such as two names. One or more of these elements can appear before or after any other element in the language. However, spaces, tabs, and comments that appear in string literals are treated as character sequences rather than delimiters.

Data Types

UIL provides literals for several of the value types it supports. Some of the value types are not supported as literals (for example, pixmaps and string tables). You can specify values for these types by using functions described in the *Functions* section. UIL directly supports the following literal types:

- String literal
- Integer literal
- Boolean literal
- Floating-point literal

UIL also includes the data type *ANY*, which is used to turn off compile time checking of data types.

String Literals

A string literal is a sequence of zero or more 8-bit or 16-bit characters or a combination delimited by ’(single quotation marks) or " (double quotation marks). String literals

can also contain multibyte characters delimited with double quotation marks. String literals can be no more than 2000 characters long.

A single-quoted string literal can span multiple source lines. To continue a single-quoted string literal, terminate the continued line with a \ (backslash). The literal continues with the first character on the next line.

Double-quoted string literals cannot span multiple source lines. (Because double-quoted strings can contain escape sequences and other special characters, you cannot use the backslash character to designate continuation of the string.) To build a string value that must span multiple source lines, use the concatenation operator described later in this section.

The syntax of a string literal is one of the following:

'[*character_string*]'
[*#char_set*]"[*character_string*]"

Both string forms associate a character set with a string value. UIL uses the following rules to determine the character set and storage format for string literals:

- A string declared as **'***string***'** is equivalent to *#cur_charset*"*string*", where *cur_charset* will be the codeset portion of the value of the *LANG* environment variable if it is set or the value of **XmFALLBACK_CHARSET** if *LANG* is not set or has no codeset component. By default, **XmFALLBACK_CHARSET** is **ISO8859-1** (equivalent to *ISO_LATIN1*), but vendors may define a different default.
- A string declared as "*string*" is equivalent to *#char_set*"*string*" if you specified *char_set* as the default character set for the module. If no default character set has been specified for the module, then if the **-s** option is provided to the **uil** command or the **use_setlocale_flag** is set for the callable compiler, **Uil()**, the string will be interpreted to be a string in the current locale. This means that the string is parsed in the locale of the user by calling *setlocale*, its charset is **XmFONTLIST_DEFAULT_TAG**, and that if the string is converted to a compound string, it is stored as a locale encoded text segment. Otherwise, "*string*" is equivalent to *#cur_charset*"*string*", where *cur_charset* is interpreted as described for single quoted strings.
- A string of the form "*string*" or *#char_set*"*string*" is stored as a null-terminated string.

If the *char_set* in a string specified in the form above is not a built-in charset, and is not a user-defined charset, the charset of the string will be set to

XmFONTLIST_DEFAULT_TAG, and an informational message will be issued to the user to note that this substitution has been made.

The following table lists the character sets supported by the UIL compiler for string literals. Note that several UIL names map to the same character set. In some cases, the UIL name influences how string literals are read. For example, strings identified by a UIL character set name ending in *_LR* are read left-to-right. Names that end in a different number reflect different fonts (for example, ISO_LATIN1 or ISO_LATIN6). All character sets in this table are represented by 8 bits.

Supported Character Sets	
UIL Name	**Description**
ISO_LATIN1	GL: ASCII, GR: Latin-1 Supplement
ISO_LATIN2	GL: ASCII, GR: Latin-2 Supplement
ISO_ARABIC	GL: ASCII, GR: Latin-Arabic Supplement
ISO_LATIN6	GL: ASCII, GR: Latin-Arabic Supplement
ISO_GREEK	GL: ASCII, GR: Latin-Greek Supplement
ISO_LATIN7	GL: ASCII, GR: Latin-Greek Supplement
ISO_HEBREW	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_LATIN8	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_HEBREW_LR	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_LATIN8_LR	GL: ASCII, GR: Latin-Hebrew Supplement
JIS_KATAKANA	GL: JIS Roman, GR: JIS Katakana

Following are the parsing rules for each of the character sets:

All character sets

Character codes in the range 00...1F, 7F, and 80...9F are control characters including both bytes of 16-bit characters. The compiler flags these as illegal characters.

ISO_LATIN1 ISO_LATIN2 ISO_LATIN3 ISO_GREEK ISO_LATIN4

These sets are parsed from left to right. The escape sequences for null-terminated strings are also supported by these character sets.

ISO_HEBREW ISO_ARABIC ISO_LATIN8

These sets are parsed from right to left. For example, the string `#ISO_HEBREW"012345"` will generate a primitive string of "543210" with character set *ISO_HEBREW*. The string direction for such a string would be right-to-left, so when rendered, the string will appear as "012345." The escape sequences for null-terminated strings are also supported by these character sets, and the characters that compose the escape sequences are in left-to-right order. For example, you would enter \n, not n\.

ISO_HEBREW_LR ISO_ARABIC_LR ISO_LATIN8_LR

These sets are parsed from left to right. For example, the string `#ISO_HEBREW_LR"012345"` generates a primitive string "012345" with character set *ISO_HEBREW*. The string direction for such a string would still be right-to-left, however, so when rendered, it will appear as "543210." In other words, the characters were originally typed in the *same order* in which they would have been typed in Hebrew (although in Hebrew, the typist would have been using a text editor that went from right to left). The escape sequences for null-terminated strings are also supported by these character sets.

JIS_KATAKANA

This set is parsed from left to right. The escape sequences for null-terminated strings are also supported by this character set. Note that the \ (backslash) may be displayed as a yen symbol.

In addition to designating parsing rules for strings, character set information remains an attribute of a compound string. If the string is included in a string consisting of several concatenated segments, the character set information is included with that string segment. This gives the Motif Toolkit the information it needs to decipher the compound string and choose a font to display the string.

For an application interface displayed only in English, UIL lets you ignore the distinctions between the two uses of strings. The compiler recognizes by context when a string must be passed as a null-terminated string or as a compound string.

The UIL compiler recognizes enough about the various character sets to correctly parse string literals. The compiler also issues errors if you use a compound string in a context that supports only null-terminated strings.

Since the character set names are keywords, you must put them in lowercase if case-sensitive names are in force. If names are case insensitive, character set names can be uppercase, lowercase, or mixed case.

In addition to the built-in character sets recognized by UIL, you can define your own character sets with the *CHARACTER_SET* function. You can use the *CHARACTER_SET* function anywhere a character set can be specified.

String literals can contain characters with the eighth (high-order) bit set. You cannot type control characters (00-1F, 7F, and 80-9F) directly in a single-quoted string literal. However, you can represent these characters with escape sequences. The following list shows the escape sequences for special characters.

\b	Backspace
\f	Form-feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\’	Single quotation mark
\""	Double quotation mark
\\	Backslash
integer\	Character whose internal representation is given by *integer* (in the range 0 to 255 decimal)

Note that escape sequences are processed literally in strings that are parsed in the current locale (localized strings).

The UIL compiler does not process newline characters in compound strings. The effect of a newline character in a compound string depends only on the character set of the string, and the result is not guaranteed to be a multiline string.

Compound String Literals

A compound string consists of a string of 8-bit, 16-bit, or multibyte characters, a named character set, and a writing direction. Its UIL data type is **compound_string**.

The writing direction of a compound string is implied by the character set specified for the string. You can explicitly set the writing direction for a compound string by using the *COMPOUND_STRING* function.

A compound string can consist of a sequence of concatenated compound strings, null-terminated strings, or a combination of both, each of which can have a different character set property and writing direction. Use the concatenation operator & (ampersand) to create a sequence of compound strings.

Each string in the sequence is stored, including the character set and writing direction information.

Generally, a string literal is stored in the UID file as a compound string when the literal consists of concatenated strings having different character sets or writing directions, or when you use the string to specify a value for an argument that requires a compound string value. If you want to guarantee that a string literal is stored as a compound string, you must use the *COMPOUND_STRING* function.

Data Storage Consumption for String Literals

The way a string literal is stored in the UID file depends on how you declare and use the string. The UIL compiler automatically converts a null-terminated string to a compound string if you use the string to specify the value of an argument that requires a compound string. However, this conversion is costly in terms of storage consumption.

PRIVATE, *EXPORTED*, and *IMPORTED* string literals require storage for a single allocation when the literal is declared; thereafter, storage is required for each reference to the literal. Literals declared in-line require storage for both an allocation and a reference.

The following table summarizes data storage consumption for string literals. The storage requirement for an allocation consists of a fixed portion and a variable portion. The fixed portion of an allocation is roughly the same as the storage requirement for a reference (a few bytes). The storage consumed by the variable portion depends on the size of the literal value (that is, the length of the string). To conserve storage space, avoid making string literal declarations that result in an allocation per use.

Data Storage Consumption for String Literals			
Declaration	**Data Type**	**Used As**	**Storage Requirements Per Use**
In-line	Null-terminated	Null-terminated	An allocation and a reference (within the module)
Private	Null-terminated	Null-terminated	A reference (within the module)
Exported	Null-terminated	Null-terminated	A reference (within the UID hierarchy)
Imported	Null-terminated	Null-terminated	A reference (within the UID hierarchy)
In-line	Null-terminated	Compound	An allocation and a reference (within the module)
Private	Null-terminated	Compound	An allocation and a reference (within the module)
Exported	Null-terminated	Compound	A reference (within the UID hierarchy)
Imported	Null-terminated	Compound	A reference (within the UID hierarchy)
In-line	Compound	Compound	An allocation and a reference (within the module)
Private	Compound	Compound	A reference (within the module)

Exported	Compound	Compound	A reference (within the UID hierarchy)
Imported	Compound	Compound	A reference (within the UID hierarchy)

Integer Literals

An integer literal represents the value of a whole number. Integer literals have the form of an optional sign followed by one or more decimal digits. An integer literal must not contain embedded spaces or commas.

Integer literals are stored in the UID file as 32-bit integers. Exported and imported integer literals require a single allocation when the literal is declared; thereafter, a few bytes of storage are required for each reference to the literal. Private integer literals and those declared in-line require allocation and reference storage per use. To conserve storage space, avoid making integer literal declarations that result in an allocation per use.

The following table shows data storage consumption for integer literals.

Data Storage Consumption for Integer Literals	
Declaration	**Storage Requirements Per Use**
In-line	An allocation and a reference (within the module)
Private	An allocation and a reference (within the module)
Exported	A reference (within the UID hierarchy)
Imported	A reference (within the UID hierarchy)

Boolean Literal

A Boolean literal represents the value True (reserved keyword **TRUE** or **On**) or False (reserved keyword **FALSE** or **Off**). These keywords are subject to case-sensitivity rules.

In a UID file, **TRUE** is represented by the integer value 1 and **FALSE** is represented by the integer value 0 (zero).

Data storage consumption for Boolean literals is the same as that for integer literals.

Floating-Point Literal

A floating-point literal represents the value of a real (or float) number. Floating-point literals have the following form:

[+|-][*integer*].*integer*[E|e[+|-]*exponent*]

For maximum portability, a floating-point literal can represent values in the range 1.0E-37 to 1.0E+37 with at least 6 significant digits. On many machines this range will be wider, with more significant digits. A floating-point literal must not contain embedded spaces or commas.

Floating-point literals are stored in the UID file as double-precision, floating-point numbers. The following table gives examples of valid and invalid floating-point notation for the UIL compiler.

Floating Point Literals	
Valid Floating-Point Literals	**Invalid Floating-Point Literals**
1.0	1e1 (no decimal point)
3.1415E−2 (equals .031415)	2.87 e6 (embedded blanks)
−6.29e7 (equals −62900000)	2.0e100 (out of range)

Data storage consumption for floating-point literals is the same as that for integer literals.

The purpose of the *ANY* data type is to shut off the data-type checking feature of the UIL compiler. You can use the *ANY* data type for the following:

- Specifying the type of a callback procedure tag
- Specifying the type of a user-defined argument

You can use the *ANY* data type when you need to use a type not supported by the UIL compiler or when you want the data-type restrictions imposed by the compiler to be relaxed. For example, you might want to define a widget having an argument that can accept different types of values, depending on run-time circumstances.

If you specify that an argument takes an *ANY* value, the compiler does not check the type of the value specified for that argument; therefore, you need to take care when specifying a value for an argument of type *ANY*. You could get unexpected results at

run time if you pass a value having a data type that the widget does not support for that argument.

Expressions

UIL includes compile-time value expressions. These expressions can contain references to other UIL values, but cannot be forward referenced.

The following table lists the set of operators in UIL that allow you to create integer, real, and Boolean values based on other values defined with the UIL module. In the table, a precedence of 1 is the highest.

Valid Operators			
Operator	**Operand Types**	**Meaning**	**Precedence**
~	Boolean	NOT	1
	integer	One's complement	
-	float	Negate	1
	integer	Negate	
+	float	NOP	1
	integer	NOP	
*	float,float	Multiply	2
	integer,integer	Multiply	
/	float,float	Divide	2
	integer,integer	Divide	
+	float,float	Add	3
	integer,integer	Add	
–	float,float	Subtract	3
	integer,integer	Subtract	
>>	integer,integer	Shift right	4
<<	integer,integer	Shift left	4
&	Boolean,Boolean	AND	5
	integer,integer	Bitwise AND	

	string,string	Concatenate	
|	Boolean,Boolean	OR	6
	integer,integer	Bitwise OR	
^	Boolean,Boolean	XOR	6
	integer,integer	Bitwise XOR	

A string can be either a single compound string or a sequence of compound strings. If the two concatenated strings have different properties (such as writing direction or character set), the result of the concatenation is a multisegment compound string.

The string resulting from the concatenation is a null-terminated string unless one or more of the following conditions exists:

- One of the operands is a compound string
- The operands have different character set properties
- The operands have different writing directions

Then the resulting string is a compound string. You cannot use imported or exported values as operands of the concatenation operator.

The result of each operator has the same type as its operands. You cannot mix types in an expression without using conversion routines.

You can use parentheses to override the normal precedence of operators. In a sequence of unary operators, the operations are performed in right-to-left order. For example, **– + –A** is equivalent to **–(+(–A))**. In a sequence of binary operators of the same precedence, the operations are performed in left-to-right order. For example, **A*B/C*D** is equivalent to **((A*B)/C)*D**.

A value declaration gives a value a name. You cannot redefine the value of that name in a subsequent value declaration. You can use a value containing operators and functions anywhere you can use a value in a UIL module. You cannot use imported values as operands in expressions.

Several of the binary operators are defined for multiple data types. For example, the operator for multiplication (*****) is defined for both floating-point and integer operands.

For the UIL compiler to perform these binary operations, both operands must be of the same type. If you supply operands of different data types, the UIL compiler automatically converts one of the operands to the type of the other according to the following conversions rules:

- If the operands are an integer and a Boolean, the Boolean is converted to an integer.
- If the operands are an integer and a floating-point, the integer is converted to an floating-point.
- If the operands are a floating-point and a Boolean, the Boolean is converted to a floating-point.

You can also explicitly convert the data type of a value by using one of the conversion functions *INTEGER*, *FLOAT* or *SINGLE_FLOAT*.

Functions

UIL provides functions to generate the following types of values:

- Character sets
- Keysyms
- Colors
- Pixmaps
- Single-precision, floating-point numbers
- Double-precision, floating-point numbers
- Fonts
- Fontsets
- Font tables
- Compound strings
- Compound string tables
- ASCIZ (null-terminated) string tables
- Wide character strings
- Widget class names
- Integer tables
- Arguments
- Reasons
- Translation tables

Remember that all examples in the following sections assume case-insensitive mode. Keywords are shown in uppercase letters to distinguish them from user-specified names, which are shown in lowercase letters. This use of uppercase letters is not required in case-insensitive mode. In case-sensitive mode, keywords must be in lowercase letters.

CHARACTER_SET(*string_expression[*, property[, ...]]**)**

You can define your own character sets with the *CHARACTER_SET* function. You can use the *CHARACTER_SET* function anywhere a character set can be specified.

The result of the *CHARACTER_SET* function is a character set with the name **string_expression** and the properties you specify. **string_expression** must be a null-terminated string. You can optionally include one or both of the following clauses to specify properties for the resulting character set:

RIGHT_TO_LEFT = *boolean_expression*
SIXTEEN_BIT = *boolean_expression*

The *RIGHT_TO_LEFT* clause sets the default writing direction of the string from right to left if *boolean_expression* is True, and right to left otherwise.

The *SIXTEEN_BIT* clause allows the strings associated with this character set to be interpreted as 16-bit characters if *boolean_expression* is True, and 8-bit characters otherwise.

KEYSYM(*string_literal***)**

The *KEYSYM* function is used to specify a keysym for a mnemonic resource. *string_literal* must contain a valid **KeySym** name. (See XStringToKeysym(3 X11) for more information.)

COLOR(string_expression[,FOREGROUND|*BACKGROUND*]**)**

The *COLOR* function supports the definition of colors. Using the *COLOR* function, you can designate a value to specify a color and then use that value for arguments requiring a color value. The string expression names the color you want to define; the optional keywords *FOREGROUND* and *BACKGROUND* identify how the color is to be displayed on a monochrome device when the color is used in the definition of a color table.

The UIL compiler does not have built-in color names. Colors are a server-dependent attribute of an object. Colors are defined on each server and may have different red-green-blue (RGB) values on each server. The string you specify as the color argument must be recognized by the server on which your application runs.

In a UID file, UIL represents a color as a character string. MRM calls X translation routines that convert a color string to the device-specific pixel value. If you are running on a monochrome server, all colors translate to black or white. If you are on a color server, the color names translate to their proper colors if the following conditions are met:

- The color is defined.
- The color map is not yet full.

If the color map is full, even valid colors translate to black or white (foreground or background).

Interfaces do not, in general, specify colors for widgets, so that the selection of colors can be controlled by the user through the **.Xdefaults** file.

To write an application that runs on both monochrome and color devices, you need to specify which colors in a color table (defined with the *COLOR_TABLE* function) map to the background and which colors map to the foreground. UIL lets you use the *COLOR* function to designate this mapping in the definition of the color. The following example shows how to use the *COLOR* function to map the color red to the background color on a monochrome device:

```
VALUE c: COLOR ( 'red',BACKGROUND );
```

The mapping comes into play only when the MRM is given a color and the application is to be displayed on a monochrome device. In this case, each color is considered to be in one of the following three categories:

- The color is mapped to the background color on the monochrome device.
- The color is mapped to the foreground color on the monochrome device.
- Monochrome mapping is undefined for this color.

If the color is mapped to the foreground or background color, MRM substitutes the foreground or background color, respectively. If you do not specify the monochrome mapping for a color, MRM passes the color string to the Motif Toolkit for mapping to the foreground or background color.

RGB(*red_integer*, *green_integer*, *blue_integer***)**

The three integers define the values for the red, green, and blue components of the color, in that order. The values of these components can range from 0 to 65,535, inclusive. The values may be represented as integer expressions.

In a UID file, UIL represents an *RGB* value as three integers. MRM calls X translation routines that convert the integers to the device-specific pixel value. If you are running on a monochrome server, all colors translate to black or white. If you are on a color server, *RGB* values translate to their proper colors if the colormap is not yet full. If the colormap is full, values translate to black or white (foreground or background).

COLOR_TABLE(*color_expression***=***'character'*[,...]**)**

The color expression is a previously defined color, a color defined in line with the *COLOR* function, or the phrase **BACKGROUND COLOR** or **FOREGROUND COLOR**. The character can be any valid UIL character.

The *COLOR_TABLE* function provides a device-independent way to specify a set of colors. The *COLOR_TABLE* function accepts either previously defined UIL color names or in line color definitions (using the *COLOR* function). A color table must be private because its contents must be known by the UIL compiler to construct an icon. The colors within a color table, however, can be imported, exported, or private.

The single letter associated with each color is the character you use to represent that color when creating an icon. Each letter used to represent a color must be unique within the color table.

ICON([**COLOR_TABLE=***color_table_name*,] *row*[,...**)**

color-table-name must refer to a previously defined color table, and *row* is a character expression giving one row of the icon.

The *ICON* function describes a rectangular icon that is x pixels wide and y pixels high. The strings surrounded by single quotation marks describe

the icon. Each string represents a row in the icon; each character in the string represents a pixel.

The first row in an icon definition determines the width of the icon. All rows must have the same number of characters as the first row. The height of the icon is dictated by the number of rows. The maximum number of rows is 999.

The first argument of the *ICON* function (the color table specification) is optional and identifies the colors that are available in this icon. By using the single letter associated with each color, you can specify the color of each pixel in the icon. The icon must be constructed of characters defined in the specified color table.

A default color table is used if you omit the argument specifying the color table. To make use of the default color table, the rows of your icon must contain only spaces and asterisks. The default color table is defined as follows:

```
COLOR_TABLE( BACKGROUND COLOR = ' ', FOREGROUND COLOR = '*')
```

You can define other characters to represent the background color and foreground color by replacing the space and asterisk in the **BACKGROUND COLOR** and **FOREGROUND COLOR** clauses shown in the previous statement. You can specify icons as private, imported, or exported. Use the MRM function **MrmFetchIconLiteral** to retrieve an exported icon at run time.

XBITMAPFILE(string_expression)

The *XBITMAPFILE* function is similar to the *ICON* function in that both describe a rectangular icon that is x pixels wide and y pixels high. However, *XBITMAPFILE* allows you to specify an external file containing the definition of an X bitmap, whereas all *ICON* function definitions must be coded directly within UIL. X bitmap files can be generated by many different X applications. UIL reads these files through the *XBITMAPFILE* function, but does not support creation of these files. The X bitmap file specified as the argument to the *XBITMAPFILE* function is read at application run time by MRM.

The *XBITMAPFILE* function returns a value of type *pixmap* and can be used anywhere a pixmap data type is expected.

SINGLE_FLOAT(*real_number_literal***)**

The *SINGLE_FLOAT* function lets you store floating-point literals in UIL files as single-precision, floating-point numbers. Single-precision floating-point numbers can often be stored using less memory than double-precision, floating-point numbers. The *real_number_literal* can be either an integer literal or a floating-point literal.

FLOAT(*real_number_literal***)**

The *FLOAT* function lets you store floating-point literals in UIL files as double-precision, floating-point numbers. The *real_number_literal* can be either an integer literal or a floating-point literal.

FONT(string_expression[, CHARACTER_SET=*char_set***])**

You define fonts with the *FONT* function. Using the *FONT* function, you designate a value to specify a font and then use that value for arguments that require a font value. The UIL compiler has no built-in fonts.

Each font makes sense only in the context of a character set. The *FONT* function has an additional parameter to let you specify the character set for the font. This parameter is optional; if you omit it, the default character set depends on the value of the *LANG* environment variable if it is set, or on the value of **XmFALLBACK_CHARSET** if *LANG* is not set.

string_expression specifies the name of the font and the clause *CHARACTER_SET = char_set* specifies the character set for the font. The string expression used in the *FONT* function cannot be a compound string.

FONTSET(string_expression[,...][**, CHARACTER_SET=***charset*])

You define fontsets with the *FONTSET* function. Using the *FONTSET* function, you designate a set of values to specify fonts and then use those values for arguments that require a fontset. The UIL compiler has no built-in fonts.

Each font makes sense only in the context of a character set. The *FONTSET* function has an additional parameter to let you specify the character set for the font. This parameter is optional; if you omit it, the default character set depends on the value of the *LANG* environment variable if it is set, or on the value of **XmFALLBACK_CHARSET** if *LANG* is not set.

The string expression specifies the name of the font and the clause *CHARACTER_SET = char_set* specifies the character set for the font. The string expression used in the *FONTSET* function cannot be a compound string.

FONT_TABLE(*font_expression*[,...]**)**

A font table is a sequence of pairs of fonts and character sets. At run time, when an object needs to display a string, the object scans the font table for the character set that matches the character set of the string to be displayed. UIL provides the *FONT_TABLE* function to let you supply such an argument. *font_expression* is created with the *FONT* and *FONTSET* functions.

If you specify a single font value to specify an argument that requires a font table, the UIL compiler automatically converts a font value to a font table.

COMPOUND_STRING(string_expression[,*property*[,...]]**)**

Use the *COMPOUND_STRING* function to set properties of a null-terminated string and to convert it into a compound string. The properties you can set are the writing direction and separator.

The result of the *COMPOUND_STRING* function is a compound string with the string expression as its value. You can optionally include one or more of the following clauses to specify properties for the resulting compound string:

RIGHT_TO_LEFT = boolean_expression SEPARATE = boolean_expression

The *RIGHT_TO_LEFT* clause sets the writing direction of the string from right to left if *boolean_expression* is True, and left to right otherwise. Specifying this argument does not cause the value of the string expression to change. If you omit the *RIGHT_TO_LEFT* argument, the resulting string has the same writing direction as **string_expression**.

The *SEPARATE* clause appends a separator to the end of the compound string if *boolean_expression* is True. If you omit the *SEPARATE* clause, the resulting string does not have a separator.

You cannot use imported or exported values as the operands of the *COMPOUND_STRING* function.

COMPOUND_STRING_COMPONENT(*component_type* [, {*string* | *enumval*}])

Use the *COMPOUND_STRING_COMPONENT* function to create compound strings in UIL consisting of single components. This function is analagous to **XmStringComponentCreate**. This function lets you create simple compound strings containing components such as **XmSTRING_COMPONENT_TAB** and **XmSTRING_COMPONENT_RENDITION_BEGIN** which are not produced by the *COMPOUND_STRING* function. These components can then be concatenated to other compound strings to build more complex compound strings.

The first argument must be an **XmStringComponentType** enumerated constant. The type and interpretation of the second argument depends on the first argument. For example, if you specify any of the following enumerated constants for the first argument, then you should not specify a second argument: **XmSTRING_COMPONENT_SEPARATOR**, **XmSTRING_COMPONENT_LAYOUT_POP**, **XmSTRING_COMPONENT_TAB**, and **XmSTRING_COMPONENT_LOCALE**. However, if you specify an enumerated constant from the following group, then you must supply a *string* as the second argument: **XmSTRING_COMPONENT_CHARSET**, **XmSTRING_COMPONENT_TEXT**, **XmSTRING_COMPONENT_LOCALE_TEXT**, **XmSTRING_COMPONENT_WIDECHAR_TEXT**, **XmSTRING_COMPONENT_RENDITION_BEGIN**, and **XmSTRING_COMPONENT_RENDITION_END**. If you specify **XmSTRING_COMPONENT_DIRECTION** as the first argument, then you must specify an **XmStringDirection** enumerated constant as the second argument. Finally, if you specify **XmSTRING_COMPONENT_LAYOUT_PUSH** as the first argument, then you must specify an **XmDirection** enumerated constant as the second argument.

The compound string components **XmSTRING_COMPONENT_RENDITION_BEGIN**, and **XmSTRING_COMPONENT_RENDITION_END** take, for their argument, the "tag," or name, of a rendition from the current render table. See the following section for more information about how to specify a render table.

COMPOUND_STRING_TABLE(string_expression[,...]**)**

A compound string table is an array of compound strings. Objects requiring a list of string values, such as the **XmNitems** and **XmNselectedItems** arguments for the list widget, use string table values. The *COMPOUND_STRING_TABLE* function builds the values for these two arguments of the list widget. The *COMPOUND_STRING_TABLE* function generates a value of type *string_table*. The name *STRING_TABLE* is a synonym for *COMPOUND_STRING_TABLE*.

The strings inside the string table must be simple strings, which the UIL compiler automatically converts to compound strings.

ASCIZ_STRING_TABLE(string_expression[,...]**)**

An ASCIZ string table is an array of ASCIZ (null-terminated) string values separated by commas. This function allows you to pass more than one ASCIZ string as a callback tag value. The *ASCIZ_STRING_TABLE* function generates a value of type **asciz_table**. The name *ASCIZ_TABLE* is a synonym for *ASCIZ_STRING_TABLE*.

WIDE_CHARACTER(string_expression)

Use the *WIDE_CHARACTER* function to generate a wide character string from null-terminated string in the current locale.

CLASS_REC_NAME(string_expression)

Use the *CLASS_REC_NAME* function to generate a widget class name. For a widget class defined by the toolkit, the string argument is the name of the class. For a user-defined widget, the string argument is the name of the creation routine for the widget.

INTEGER_TABLE(*integer_expression*[,...]**)**

An integer table is an array of integer values separated by commas. This function allows you to pass more than one integer per callback tag value. The *INTEGER_TABLE* function generates a value of type **integer_table**.

ARGUMENT(string_expression[, *argument_type*]**)**

The *ARGUMENT* function defines the arguments to a user-defined widget. Each of the objects that can be described by UIL permits a set of arguments, listed in Appendix B. For example, **XmNheight** is an argument to most objects and has an integer data type. To specify height for a user-defined widget, you can use the built-in argument name **XmNheight**, and specify an integer value when you declare the user-

defined widget. You do not use the *ARGUMENT* function to specify arguments that are built into the UIL compiler.

The **string_expression** name is the name the UIL compiler uses for the argument in the UID file. *argument_type* is the type of value that can be associated with the argument. If you omit the second argument, the default type is *ANY* and no value type checking occurs. Use one of the following keywords to specify the argument type:

- ANY
- ASCIZ_TABLE
- BOOLEAN
- COLOR
- COMPOUND_STRING
- FLOAT
- FONT
- FONT_TABLE
- FONTSET
- ICON
- INTEGER
- INTEGER_TABLE
- KEYSYM
- PIXMAP
- REASON
- SINGLE_FLOAT
- STRING
- STRING_TABLE
- TRANSLATION_TABLE
- WIDE_CHARACTER
- WIDGET

You can use the *ARGUMENT* function to allow the UIL compiler to recognize extensions to the Motif Toolkit. For example, an existing widget may accept a new argument. Using the *ARGUMENT* function, you can make this new argument available to the UIL compiler before the updated version of the compiler is released.

REASON(string_expression)

The *REASON* function is useful for defining new reasons for user-defined widgets.

Each of the objects in the Motif Toolkit defines a set of conditions under which it calls a user-defined function. These conditions are known as callback reasons. The user-defined functions are termed callback procedures. In a UIL module, you use a callbacks list to specify which user-defined functions are to be called for which reasons.

Appendix B lists the callback reasons supported by the Motif Toolkit objects.

When you declare a user-defined widget, you can define callback reasons for that widget using the *REASON* function. The string expression specifies the argument name stored in the UID file for the reason. This reason name is supplied to the widget creation routine at run time.

TRANSLATION_TABLE(string_expression[,...]**)**

Each of the Motif Toolkit widgets has a translation table that maps X events (for example, mouse button 1 being pressed) to a sequence of actions. Through widget arguments, such as the common translations argument, you can specify an alternate set of events or actions for a particular widget. The *TRANSLATION_TABLE* function creates a translation table that can be used as the value of an argument that is of the data type **translation_table**.

You can use one of the following translation table directives with the *TRANSLATION_TABLE* function: **#override**, **#augment**, or **#replace**. The default is **#replace**. If you specify one of these directives, it must be the first entry in the translation table.

The **#override** directive causes any duplicate translations to be ignored. For example, if a translation for <**Btn1Down**> is already defined in the current translations for a PushButton, the translation defined by *new_translations* overrides the current definition. If the **#augment** directive is specified, the current definition takes precedence. The

#replace directive replaces all current translations with those specified in the **XmNtranslations** resource.

Renditions and Render Tables

In addition to the string direction, each compound string carries a great deal of information about how its text is to be rendered. Each compound string contains a "tag," identifying the "rendition" to be used to draw that string. The rendition contains such information as the font, the size, the color, whether the text is to be underlined or crossed out, and the position and style of any tab stops. Many renditions are combined into a "render table," which is specified to any widget with the **XmNrenderTable** resource, and in the widget's *controls* list.

UIL implements render tables, renditions, tab lists, and tab stops as a special class of objects, in a form similar to the widget class. These objects are not themselves widgets or gadgets, but the format used by UIL to specify widget resources provides a convenient way to specify the qualities and dependencies of these objects.

For example, a render table, included in some widget's *controls* list, must also have a *controls* list in its specification, containing the names of its member renditions. Each rendition, in its specification, will contain an *arguments* list specifying such qualities as the font, the color, and whether the text is to be underlined. Any of the renditions may also control a tablist, which will itself control one or more tab stops.

Please refer to the *Motif 2.1—Programmer's Guide* for a complete description of renditions and render tables, and for an example of how to use them in UIL.

Related Information

uil(1), **Uil**(3)

WML

Purpose

The widget meta-language file format for creating uil compilers

Description

The widget meta-language facility (WML) is used to generate the components of the user interface language (UIL) compiler that can change depending on the widget set. Using WML you can add support in UIL for new widgets to the Motif widget set or for a totally new widget set.

File

WML files are ASCII files that you can modify with any standard text editor. They are accessed in the **tools/wml** directory by WML. By convention WML files have the suffix **.wml**. The Motif widget set is described in the **motif.wml** file. This is also the default WML file when using the WML facility.

When adding new widgets or changing widget characteristics, you should start with a copy of the **motif.wml** file. If you are creating a new widget set for use with UIL, you should start from scratch. In either case the **motif.wml** file is a good example of WML syntax, and you should familiarize yourself with it before writing your own WML file.

WML files have a simple syntax, similar in structure to UIL. It is made up of the following elements:

- Comments
- Data Type Definitions
- Character Set Definitions
- Enumeration Set Definitions
- Control List Definitions
- Class Definitions
- Child Definitions
- Resource Definitions

You can use space, tabs, or newlines anywhere in the syntax, as long as you do not split up keywords or strings, except that comments end at a newline. The order of elements is not important to the syntax.

This description uses the following additional conventions to describe the syntax of the widget meta-language:

[ ] Indicates optional elements.

... Indicates where an element of syntax can be repeated.

| Indicates a choice among multiple items.

Comments

You can include comments in the WML file. Comments have the following syntax:

[any.element]!any.comment

Comments begin with an exclamation point and extend to the end of the line. A comment can begin on a line by itself or follow any part of another element. A comment does not change the meaning of any other element. For example:

```
!This is a comment
!  that spans two lines.
DataType    !This is a comment following code.
```

Data Type Definitions

Data type definitions register all the resource data types used in the file. You must register all the data types used in your WML file. Data type definitions have the following syntax:

DataType
    any.datatype [{ InternalLiteral = internal.name |
        DocName = "*string*"; [...]}];
    [...]

A data type definition begins with the keyword **DataType**. Following the **DataType** keyword is a list of data types that can be further modified with

InternalLiteral
This forces the value of the internal symbol table literal definition of the data type name. This modifier is only used to get around symbol table definitions hard coded into the UIL compiler. It should rarely be used.

DocName This gives an arbitrary string for use in the documentation. This string is meant to supply a different name for the data type for use in the documentation, or a single name for the data type if the data type has aliases.

For example:

```
DataType OddNumber {DocName="OddNumber";};
         NewString;
```

Character Set Definitions

Character set definitions register the Motif Toolkit name and other information for the character set names used in UIL. Character set definitions have the following syntax:

```
CharacterSet
    any.character.set
        { [ FontListElementTag | XmStringCharsetName ] = "string";
            [ Alias = "string" ...; |
            Direction = [ LeftToRight | RightToLeft ]; |
            ParseDirection = [ LeftToRight | RightToLeft ]; |
            CharacterSize = [ OneByte | TwoByte ]; ]
            [ ... ] };
    [ ... ]
```

A character set definition begins with the keyword **CharacterSet**. Following the **CharacterSet** keyword is a list of character sets that can be further modified with

FontListElementTag | *XmStringCharsetName*
Specifies the name of the character set, which will become the character set component of a compound string segment created using this character set. This modifier is required.

Alias Specifies one or more aliases for the character set name. Each alias can be used within UIL to refer to the same character set.

Direction Specifies the direction of a compound string segment created using this character set. The default is **LeftToRight**.

ParseDirection
Specifies the direction in which an input string is parsed when a compound string segment is created using this character set. The default is whatever **Direction** is specified.

CharacterSize

Specifies the number of bytes in each character of a compound string segment created using this character set. The default is **OneByte**.

For example:

```
CharacterSet
  iso_latin1
    { XmStringCharsetName = "ISO8859-1";
      Alias = "ISOLatin1"; };
  iso_hebrew_lr
    { XmStringCharsetName = "ISO8859-8";
      Alias = "iso_latin8_lr";
      Direction = RightToLeft;
      ParseDirection = LeftToRight; };
  ksc_korean
    { XmStringCharsetName = "KSC5601.1987-0";
      CharacterSize = TwoByte; };
```

Enumeration Set Definitions

Enumeration set definitions register the named constants used in the Motif Toolkit to specify some resource values. Enumeration set definitions have the following syntax:

```
EnumerationSet
    resource.name: resource.type
        { enum.value.name; [ ... ] };
```

An enumeration set definition begins with the keyword **EnumerationSet**. For each enumeration set defined, the name and type of the resource are listed. The resource name is the Motif Toolkit resource name, with the beginning **XmN** removed and with the initial letter capitalized. For example, the name of the Motif Toolkit resource **XmNrowColumnType** is **RowColumnType**. The resource type is the data type for the resource; for most resources, this is *integer*. Following the resource name and type is a list of names of enumeration values that can be used as settings for the resource. These names are the same as those in the Motif Toolkit.

For example:

```
EnumerationSet
  RowColumnType: integer
    { XmWORK_AREA; XmMENU_BAR; XmMENU_POPUP;
```

```
XmMENU_PULLDOWN; XmMENU_OPTION; };
```

Enumeration sets also support Boolean values.

Control List Definitions

Control list definitions assign a name to groups of controls. You can use these control lists later in class definitions to simplify the structure of your WML file. Control list definitions have the following syntax:

ControlList
    any.control.list [{ any.control; [...]}];

A control list definition starts with the **ControlList** keyword. Following the **ControlList** keyword are any number of control list definitions. Control list definitions are made up of a control list name followed by the set of controls it represents. For example:

```
ControlList
        Buttons {PushButton;
                 RadioButton;
                 CascadeButton;
                 NewCascadebutton;};
```

Each control specified in the control list must be defined as a class in the file.

Class Definitions

Class definitions describe a particular widget class including its position in the class hierarchy, toolkit convenience function, resources, and controls. There should be one class definition for each widget or gadget in the widget set you want to support in UIL. Class definitions have the following syntax:

Class class.name: MetaClass | Widget | Gadget
    [{[
    SuperClass = class.name; |
    ParentClass = parent.class.name; |
    InternalLiteral = internal.name; |
    Alias = *alias*; |
    ConvenienceFunction = convenience.function; |
    WidgetClass = widget.class; |
    DocName = "*string*"; |
    DialogClass = True | False; |

```
Resources { any.resource.name [{
        Default = new.default.value; |
        Exclude = True |
        False;
        [...]} ];
    [...]}; |
Controls { any.control.name; [...]};
Children { any.child.name; [...] };
[...]
]}];
```

Class definitions start with the **Class** keyword. For each class defined, the name of the class and whether the class is a metaclass, widget, or gadget is listed. Each class definition can be further modified with the keywords described in the following list.

SuperClass This indicates the name of the parent class. Only the root of the hierarchy does not specify a SuperClass.

ParentClass This indicates the name of the widget's automatically created parent class if one exists. This allows resources for that automatically created class to be used in instances of this class. For example, *XmBulletinBoardDialog* creates both an **XmBulletinBoard** and an **XmDialogShell**. To access the resources of the **XmDialogShell** parent class it must be specified here.

InternalLiteral
This forces the value of the internal symbol table literal definition of the class name. This modifier is only used to get around symbol table definitions hard coded into the UIL compiler. It should rarely be used.

Alias This indicates alternate names for the class for use in a UIL specification.

ConvenienceFunction
This indicates the name of the creation convenience function for this class. All widget and gadget classes must have a **ConvenienceFunction.**

WidgetClass
This indicates the associated widget class of gadget type classes. Presently, nothing is done with this value.

DocName This defines an arbitrary string for use in the documentation. Presently, nothing is done with this value.

DialogClass This indicates whether the class is a dialog class. Presently, nothing is done with this value.

Resources This lists the resources of the widget class. This keyword can be further modified with

Default This specifies a new default value for this resource. Resource default values are usually set in the resource definition. If an inherited resource's default value is changed by the class, the new default value should be noted here.

Exclude This specifies whether an inherited resource should be excluded from the resource list of the class. **Exclude** is False by default.

Children This lists the names of the automatically created children of this class, so that those children can be accessed in the UIL file.

Controls This lists the controls that the widget class allows. The controls can be other classes or a control list from the control list definition.

The following example uses the examples from the data type definitions and control list definitions above.

```
Class
     TopLevelWidget: MetaClass
          {
          Resources
               {
               XtbNfirstResource;
               XtbNsecondResource;
               };
          };
     NewWidget: Widget
          {
          SuperClass = TopLevelWidget;
          ConvenienceFunction =
              XtbCreateNewWidget;
          Resources
               {
               XtbNnewResource;
               XtbNfirstResource
```

```
            {Default="XtbNEW_VALUE";};
        XtbNsecondResource
            {Exclude=True;};
        };
    Controls
        {
        NewWidget;
        Buttons;
        };
};
```

Child Definitions

Child definitions register the classes of automatically created children. Automatically created children are referenced elsewhere in a **uil** file using the **Children** keyword within a class definition. Child definitions have the following syntax:

Child child.name : class.name; [...]

Where **child.name** is the name of the automatically created child and **class.name** is the name of the class of that child.

Resource Definitions

Resource definitions describe a particular resource including its type, and default value. There should be a resource definition for each new resource referenced in the class definitions. Resource definitions have the following syntax:

Resource
    resource.name: Argument | Reason | Constraint | SubResource
        [{[
        Type = *type*;
        [ResourceLiteral = resource.literal;]
        [InternalLiteral = internal.name;]
        [Alias = *alias*;]
        [Related = *related*;]
        [Default = *default*;]
        [DocName = doc.name;]
        [...]}]
    [...]

Resource definitions start with the **Resource** keyword. For each resource definition, the name of the resource and whether the resource is an argument, reason, constraint or subresource is listed.

Argument Indicates a standard resource

Reason Indicates a callback resource

Constraint Indicates a constraint resource

SubResource
Presently, nothing is done with this value

The resource definition can be further modified with the following keywords:

Type This indicates the data type of the resource. It must be listed in the data type definition.

ResourceLiteral
This indicates the keyword used in the UIL file to reference the resource. In Motif, the resource name is the same as the **ResourceLiteral**.

InternalLiteral
This forces the value of the internal symbol table literal definition of the resource name. This modifier is only used to get around symbol table definitions hard coded into the UIL compiler. It should rarely be used.

Alias This indicates alternate names for the resource for use in a UIL specification.

Related This is a special purpose field that allows resources that act as a counter for the current resources to be related to the resource. UIL automatically sets the value of this related resource to the number of items in the compiled instance of type **resource.name**.

Default This indicates the default value of the resource.

DocName This defines an arbitrary string for use in the documentation. Presently, nothing is done with this value.

The following example uses the examples from the data type definitions, control list definitions and class definitions above.

```
Resource
     XtbNfirstResource: Argument
          { Type = OddNumber;
```

```
       Default = "XtbOLD_VALUE";};
XtbNsecondResource: Argument
     { Type = NewString;
       Default = "XtbNEW_STRING"; };
XtbNnewResource: Argument
     { Type = OddNumber;
       Default = "XtbODD_NUMBER"; };
```

Appendix A

Constraint Arguments and Automatically Created Children

The following tables list the constraint arguments and automatically created children for widgets available within UIL. The constraints are available for children of the listed widget. For more information about constraint arguments see the *Motif 2.1—Programmer's Guide*.

XmForm and XmFormDialog Constraint Arguments	
XmNbottomAttachment	XmNrightAttachment
XmNbottomOffset	XmNrightOffset
XmNbottomPosition	XmNrightPosition
XmNbottomWidget	XmNrightWidget
XmNleftAttachment	XmNtopAttachment
XmNleftOffset	XmNtopOffset
XmNleftPosition	XmNtopPosition

XmNleftWidget	XmNtopWidget
XmNresizable	

XmFrame Constraint Arguments	
XmNchildHorizontalAlignment	XmNframechildType
XmNchildHorizontalSpacing	XmNchildVerticalAlignment

XmPanedWindow Constraint Arguments	
XmNallowResize	XmNpositionIndex
XmNpaneMaximum	XmNskipAdjust
XmNpaneMinimum	

XmRowColumn Constraint Arguments	
XmNpositionIndex	

XmScrolledWindow Constraint Arguments	
XmNscrolledWindowChildType	

XmSelectionBox Constraint Arguments	
XmNchildPlacement	

XmCommand Automatically Created Children
XmCommand inherits its automatically created children from
XmSelectionBox

XmFileSelectionBox Automatically Created Children	
Name	**Class**
Xm_Items	XmLabelGadget
Xm_ItemsList	XmScrolledList

Xm_Separator	XmSeparatorGadget
Xm_OK	XmPushButtonGadget
Xm_Cancel	XmPushButtonGadget
Xm_Help	XmPushButtonGadget
Xm_FilterLabel	XmLabelGadget
Xm_FilterText	XmText
Xm_DirList	XmScrolledList
Xm_Dir	XmLabelGadget
Xm_Filter	XmPushButtonGadget
XmFileSelectionBox also inherits its automatically created	
children from **XmSelectionBox**	

XmMainWindow Automatically Created Children	
Name	**Class**
Xm_Separator1	XmSeparatorGadget
Xm_Separator2	XmSeparatorGadget
Xm_Separator3	XmSeparatorGadget
XmMainWindow also inherits its automatically created children	
from **XmScrolledWindow**	

XmMessageBox Automatically Created Children	
Name	**Class**
Xm_Symbol	XmLabelGadget
Xm_Separator	XmSeparatorGadget
Xm_Message	XmLabelGadget
Xm_OK	XmPushButtonGadget
Xm_Cancel	XmPushButtonGadget
Xm_Help	XmPushButtonGadget

XmOptionMenu Automatically Created Children	
Name	**Class**
Xm_OptionLabel	XmLabelGadget
Xm_OptionButton	XmCascadeButtonGadget

XmPanedWindow Automatically Created Children	
Name	**Class**
Xm_Sash	undocumented subclass of XmPrimitive
Xm_Separator	XmSeparatorGadget

XmPopup and XmPulldownMenu Automatically Created Children	
Name	**Class**
Xm_TearOffControl	undocumented subclass of XmPushButton

XmScale Automatically Created Children	
Name	**Class**
Xm_Scrollbar	XmScrollBar
Xm_Title	XmLabelGadget

XmScrolledWindow Automatically Created Children	
Name	**Class**
Xm_ClipWindow	XmClipWindow
Xm_VertScrollBar	XmScrollBar
Xm_HorScrollBar	XmScrollBar

XmSelectionBox Automatically Created Children	
Name	**Class**
Xm_Items	XmLabelGadget

Xm_ItemsList	XmScrolledList
Xm_Selection	XmLabelGadget
Xm_Text	XmText
Xm_Separator	XmSeparatorGadget
Xm_OK	XmPushButtonGadget
Xm_Cancel	XmPushButtonGadget
Xm_Help	XmPushButtonGadget
Xm_Apply	XmPushButtonGadget

Appendix B

UIL Built-In Tables

This appendix contains a listing of part of the UIL built-in tables used during compilation to check that your UIL specification is consistent with the Motif Toolkit.

For each object in the Motif Toolkit, this appendix contains a table that lists the reasons and controls (children) supported by UIL for that object. The arguments supported by UIL for each object are the same as the Motif Toolkit resources for that object. Appendix C lists the name and UIL data type of each UIL argument. For information on which arguments are supported for which objects and for the default values of arguments, see the widget reference pages.

XmArrowButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNconvertCallback

XmArrowButton	
Controls	**Reasons**
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback
	XmNpopupHandlerCallback

XmArrowButtonGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback

XmBulletinBoard	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNpopupHandlerCallback
XmCheckBox	XmNunmapCallback
XmComboBox	XmNlosingFocusCallback
XmCommand	
XmCommandDialog	

XmBulletinBoard	
Controls	**Reasons**
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	

XmBulletinBoard	
Controls	**Reasons**
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	

XmBulletinBoard	
Controls	**Reasons**
XmWorkingDialog	
user_defined	

XmBulletinBoardDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNfocusMovedCallback
XmCascadeButton	XmNlosingFocusCallback
XmCascadeButtonGadget	XmNmapCallback
XmCheckBox	XmNpopdownCallback
XmComboBox	XmNpopupCallback
XmCommand	XmNpopupHandlerCallback
XmCommandDialog	XmNrealizeCallback
XmContainer	XmNunmapCallback
XmDialogShell	XmNhelpCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	

XmBulletinBoardDialog	
Controls	**Reasons**
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	

XmBulletinBoardDialog	
Controls	**Reasons**
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmCascadeButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
XmPulldownMenu	XmNactivateCallback
	XmNcascadingCallback
	XmNconvertCallback
	XmNdestroyCallback
	XmNhelpCallback
	XmNpopupHandlerCallback

XmCascadeButtonGadget	
Controls	**Reasons**
XmPulldownMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNcascadingCallback
	XmNdestroyCallback
	XmNhelpCallback

XmCheckBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNpopupHandlerCallback
XmCascadeButtonGadget	XmNsimpleCallback
XmCheckBox	XmNtearOffMenuActivateCallback
XmComboBox	XmNtearOffMenuDeactivateCallback
XmCommand	XmNunmapCallback
XmCommandDialog	XmNmapCallback
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	

XmCheckBox	
Controls	**Reasons**
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	

XmCheckBox	
Controls	**Reasons**
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmComboBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNpopupHandlerCallback
XmCascadeButton	XmNselectionCallback

XmComboBox	
Controls	**Reasons**
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	

XmComboBox	
Controls	**Reasons**
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	

XmComboBox	
Controls	**Reasons**
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmCommand	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNcommandChangedCallback
	XmNcommandEnteredCallback
	XmNdestroyCallback
	XmNfocusCallback
	XmNhelpCallback
	XmNlosingFocusCallback
	XmNmapCallback
	XmNpopupHandlerCallback
	XmNunmapCallback

XmCommandDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcommandChangedCallback
XmBulletinBoard	XmNcommandEnteredCallback
XmBulletinBoardDialog	XmNdestroyCallback

XmCommandDialog	
Controls	**Reasons**
XmCascadeButton	XmNfocusMovedCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCheckBox	XmNlosingFocusCallback
XmComboBox	XmNmapCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	

XmCommandDialog	
Controls	**Reasons**
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	

XmCommandDialog	
Controls	**Reasons**
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmContainer	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNconvertCallback
XmBulletinBoard	XmNdefaultActionCallback
XmBulletinBoardDialog	XmNdestinationCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNoutlineChangedCallback
XmCheckBox	XmNpopupHandlerCallback
XmComboBox	XmNselectionCallback
XmCommand	XmNdestroyCallback
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	

XmContainer	
Controls	**Reasons**
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	

XmContainer	
Controls	**Reasons**
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmDialogShell	
Controls	**Reasons**
XmBulletinBoard	MrmNcreateCallback
XmCheckBox	XmNdestroyCallback
XmComboBox	XmNfocusMovedCallback

XmDialogShell	
Controls	**Reasons**
XmContainer	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmFileSelectionBox	XmNrealizeCallback
XmForm	
XmFrame	
XmMessageBox	
XmNotebook	
XmPanedWindow	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrolledWindow	
XmSelectionBox	
XmSpinBox	
XmWorkArea	

XmDrawingArea	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNconvertCallback
XmBulletinBoard	XmNdestinationCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNinputCallback
XmCheckBox	XmNpopupHandlerCallback
XmComboBox	XmNresizeCallback

XmDrawingArea	
Controls	**Reasons**
XmCommand	XmNexposeCallback
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	

XmDrawingArea	
Controls	**Reasons**
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	

XmDrawingArea	
Controls	**Reasons**
XmWorkArea	
XmWorkingDialog	
user_defined	

XmDrawnButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNconvertCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNexposeCallback
	XmNhelpCallback
	XmNpopupHandlerCallback
	XmNresizeCallback

XmErrorDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback
XmCheckBox	XmNmapCallback

XmErrorDialog	
Controls	**Reasons**
XmComboBox	XmNokCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusMovedCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	

XmErrorDialog	
Controls	**Reasons**
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
ScrolledList	
XmXmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	

XmErrorDialog	
Controls	**Reasons**
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFileSelectionBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback
XmCheckBox	XmNmapCallback
XmComboBox	XmNnoMatchCallback
XmCommand	XmNokCallback
XmCommandDialog	XmNpopupHandlerCallback
XmContainer	XmNunmapCallback
XmDialogShell	XmNfocusCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	

XmFileSelectionBox	
Controls	**Reasons**
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	

XmFileSelectionBox	
Controls	**Reasons**
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFileSelectionDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusMovedCallback
XmCascadeButtonGadget	XmNhelpCallback

XmFileSelectionDialog	
Controls	**Reasons**
XmCheckBox	XmNlosingFocusCallback
XmComboBox	XmNmapCallback
XmCommand	XmNnoMatchCallback
XmCommandDialog	XmNokCallback
XmContainer	XmNpopdownCallback
XmDialogShell	XmNpopupCallback
XmDrawingArea	XmNpopupHandlerCallback
XmDrawnButton	XmNrealizeCallback
XmErrorDialog	XmNunmapCallback
XmFileSelectionBox	XmNfocusCallback
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	

XmFileSelectionDialog	
Controls	**Reasons**
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	

XmFileSelectionDialog	
Controls	**Reasons**
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmForm	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNpopupHandlerCallback
XmCheckBox	XmNunmapCallback
XmComboBox	XmNlosingFocusCallback
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	

XmForm	
Controls	**Reasons**
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	

XmForm	
Controls	**Reasons**
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFormDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNfocusMovedCallback
XmCascadeButton	XmNlosingFocusCallback

XmFormDialog	
Controls	**Reasons**
XmCascadeButtonGadget	XmNmapCallback
XmCheckBox	XmNpopdownCallback
XmComboBox	XmNpopupCallback
XmCommand	XmNpopupHandlerCallback
XmCommandDialog	XmNrealizeCallback
XmContainer	XmNunmapCallback
XmDialogShell	XmNhelpCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	

XmFormDialog	
Controls	**Reasons**
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	

XmFormDialog	
Controls	**Reasons**
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFrame	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNpopupHandlerCallback
XmCascadeButton	
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	

XmFrame	
Controls	**Reasons**
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	

XmFrame	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmIconGadget	
Controls	**Reasons**
XmRenderTable	MrmNcreateCallback
	XmNdestroyCallback
	XmNhelpCallback

XmInformationDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback
XmCheckBox	XmNmapCallback
XmComboBox	XmNokCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusMovedCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	

XmInformationDialog	
Controls	**Reasons**
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	

XmInformationDialog	
Controls	**Reasons**
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmLabel	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
XmRenderTable	XmNconvertCallback
	XmNdestroyCallback
	XmNhelpCallback
	XmNpopupHandlerCallback

XmLabelGadget	
Controls	**Reasons**
XmRenderTable	MrmNcreateCallback
	XmNdestroyCallback
	XmNhelpCallback

XmList	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
XmRenderTable	XmNbrowseSelectionCallback
	XmNconvertCallback
	XmNdefaultActionCallback
	XmNdestinationCallback
	XmNdestroyCallback
	XmNextendedSelectionCallback
	XmNhelpCallback
	XmNmultipleSelectionCallback
	XmNpopupHandlerCallback
	XmNsingleSelectionCallback

XmMainWindow	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNpopupHandlerCallback
XmCascadeButton	XmNtraverseObscuredCallback
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	

XmMainWindow	
Controls	**Reasons**
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	

XmMainWindow	
Controls	**Reasons**
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmMenuBar	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNdestroyCallback
XmDrawnButton	XmNentryCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmapCallback
XmPopupMenu	XmNpopupHandlerCallback
XmPulldownMenu	XmNtearOffMenuActivateCallback
XmPushButton	XmNtearOffMenuDeactivateCallback
XmPushButtonGadget	XmNunmapCallback
XmSeparator	
XmSeparatorGadget	
XmToggleButton	
XmToggleButtonGadget	
user_defined	

XmMenuShell	
Controls	**Reasons**
XmRenderTable	MrmNcreateCallback
XmRowColumn	XmNdestroyCallback
	XmNpopdownCallback
	XmNpopupCallback

XmMessageBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback

XmMessageBox	
Controls	**Reasons**
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNlosingFocusCallback
XmCascadeButtonGadget	XmNmapCallback
XmCheckBox	XmNokCallback
XmComboBox	XmNpopupHandlerCallback
XmCommand	XmNunmapCallback
XmCommandDialog	XmNhelpCallback
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	

XmMessageBox	
Controls	**Reasons**
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	

XmMessageBox	
Controls	**Reasons**
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmMessageDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback
XmCheckBox	XmNmapCallback
XmComboBox	XmNokCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusMovedCallback

XmMessageDialog	
Controls	**Reasons**
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	

XmMessageDialog	
Controls	**Reasons**
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmNotebook	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNpageChangedCallback
XmCascadeButton	XmNpopupHandlerCallback
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	

XmNotebook	
Controls	**Reasons**
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	

XmNotebook	
Controls	**Reasons**
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmOptionMenu	
Controls	**Reasons**
XmPulldownMenu	MrmNcreateCallback
	XmNdestroyCallback
	XmNentryCallback
	XmNhelpCallback
	XmNmapCallback
	XmNpopupHandlerCallback
	XmNtearOffMenuActivateCallback
	XmNtearOffMenuDeactivateCallback
	XmNunmapCallback

XmPanedWindow	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNpopupHandlerCallback
XmCascadeButton	
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	

XmPanedWindow	
Controls	**Reasons**
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	

XmPanedWindow	
Controls	**Reasons**
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmPopupMenu	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNdestroyCallback
XmDrawnButton	XmNentryCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmapCallback
XmPushButton	XmNpopdownCallback
XmPushButtonGadget	XmNpopupCallback
XmSeparator	XmNpopupHandlerCallback
XmSeparatorGadget	XmNtearOffMenuActivateCallback
XmToggleButton	XmNtearOffMenuDeactivateCallback
XmToggleButtonGadget	XmNunmapCallback
user_defined	

XmPromptDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusMovedCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCheckBox	XmNlosingFocusCallback
XmComboBox	XmNmapCallback
XmCommand	XmNnoMatchCallback
XmCommandDialog	XmNokCallback
XmContainer	XmNpopdownCallback
XmDialogShell	XmNpopupCallback
XmDrawingArea	XmNpopupHandlerCallback
XmDrawnButton	XmNrealizeCallback
XmErrorDialog	XmNunmapCallback
XmFileSelectionBox	XmNfocusCallback
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	

XmPromptDialog	
Controls	**Reasons**
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	

XmPromptDialog	
Controls	**Reasons**
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmPulldownMenu	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNdestroyCallback
XmDrawnButton	XmNentryCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmapCallback
XmPushButton	XmNpopdownCallback
XmPushButtonGadget	XmNpopupCallback
XmSeparator	XmNpopupHandlerCallback
XmSeparatorGadget	XmNtearOffMenuActivateCallback
XmToggleButton	XmNtearOffMenuDeactivateCallback
XmToggleButtonGadget	XmNunmapCallback
user_defined	

XmPushButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNconvertCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback
	XmNpopupHandlerCallback

XmPushButtonGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback

XmQuestionDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback

XmQuestionDialog	
Controls	**Reasons**
XmCheckBox	XmNmapCallback
XmComboBox	XmNokCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusMovedCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	

XmQuestionDialog	
Controls	**Reasons**
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	

XmQuestionDialog	
Controls	**Reasons**
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmRadioBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNpopupHandlerCallback
XmCascadeButtonGadget	XmNtearOffMenuActivateCallback
XmCheckBox	XmNtearOffMenuDeactivateCallback
XmComboBox	XmNunmapCallback
XmCommand	XmNmapCallback
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	

XmRadioBox	
Controls	**Reasons**
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	

XmRadioBox	
Controls	**Reasons**
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmRenderTable	
Controls	**Reasons**
XmRendition	MrmNcreateCallback
	XmNdestroyCallback

XmRendition	
Controls	**Reasons**
XmTabList	MrmNcreateCallback
	XmNdestroyCallback

XmRowColumn	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNpopupHandlerCallback
XmCascadeButtonGadget	XmNtearOffMenuActivateCallback
XmCheckBox	XmNtearOffMenuDeactivateCallback
XmComboBox	XmNunmapCallback
XmCommand	XmNmapCallback
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	

XmRowColumn	
Controls	**Reasons**
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	

XmRowColumn	
Controls	**Reasons**
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmScale	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNconvertCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNdragCallback
XmCascadeButton	XmNpopupHandlerCallback
XmCascadeButtonGadget	XmNvalueChangedCallback
XmCheckBox	XmNhelpCallback
XmComboBox	

XmScale	
Controls	**Reasons**
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	

XmScale	
Controls	**Reasons**
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	

XmScale	
Controls	**Reasons**
XmWorkArea	
XmWorkingDialog	
user_defined	

XmScrollBar	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNconvertCallback
	XmNdecrementCallback
	XmNdestroyCallback
	XmNdragCallback
	XmNhelpCallback
	XmNincrementCallback
	XmNpageDecrementCallback
	XmNpageIncrementCallback
	XmNpopupHandlerCallback
	XmNtoBottomCallback
	XmNtoTopCallback
	XmNvalueChangedCallback

XmScrolledList	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNbrowseSelectionCallback
	XmNconvertCallback
	XmNdefaultActionCallback

XmScrolledList	
Controls	**Reasons**
	XmNdestinationCallback
	XmNdestroyCallback
	XmNextendedSelectionCallback
	XmNhelpCallback
	XmNmultipleSelectionCallback
	XmNpopupHandlerCallback
	XmNsingleSelectionCallback
	XmNtraverseObscuredCallback

XmScrolledText	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNconvertCallback
	XmNdestinationCallback
	XmNdestroyCallback
	XmNfocusCallback
	XmNgainPrimaryCallback
	XmNhelpCallback
	XmNlosePrimaryCallback
	XmNlosingFocusCallback
	XmNmodifyVerifyCallback
	XmNmodifyVerifyCallbackWcs
	XmNmotionVerifyCallback
	XmNpopupHandlerCallback

XmScrolledText	
Controls	**Reasons**
	XmNtraverseObscuredCallback
	XmNvalueChangedCallback

XmScrolledWindow	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNpopupHandlerCallback
XmCascadeButton	XmNtraverseObscuredCallback
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	

XmScrolledWindow	
Controls	**Reasons**
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	

XmScrolledWindow	
Controls	**Reasons**
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmSelectionBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback
XmCheckBox	XmNmapCallback
XmComboBox	XmNnoMatchCallback

XmSelectionBox	
Controls	**Reasons**
XmCommand	XmNokCallback
XmCommandDialog	XmNpopupHandlerCallback
XmContainer	XmNunmapCallback
XmDialogShell	XmNfocusCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	

XmSelectionBox	
Controls	**Reasons**
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	

XmSelectionBox	
Controls	**Reasons**
XmWorkArea	
XmWorkingDialog	
user_defined	

XmSelectionDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusMovedCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCheckBox	XmNlosingFocusCallback
XmComboBox	XmNmapCallback
XmCommand	XmNnoMatchCallback
XmCommandDialog	XmNokCallback
XmContainer	XmNpopdownCallback
XmDialogShell	XmNpopupCallback
XmDrawingArea	XmNpopupHandlerCallback
XmDrawnButton	XmNrealizeCallback
XmErrorDialog	XmNunmapCallback
XmFileSelectionBox	XmNfocusCallback
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	

XmSelectionDialog	
Controls	**Reasons**
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	

XmSelectionDialog	
Controls	**Reasons**
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmSeparator	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNconvertCallback
	XmNdestroyCallback
	XmNhelpCallback
	XmNpopupHandlerCallback

XmSeparatorGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback
	XmNdestroyCallback
	XmNhelpCallback

XmSimpleSpinBox	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNactivateCallback
XmDrawnButton	XmNdestroyCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmodifyVerifyCallback
XmPushButton	XmNpopupHandlerCallback
XmPushButtonGadget	XmNvalueChangedCallback
XmSeparator	XmNlosingFocusCallback
XmSeparatorGadget	
XmToggleButton	
XmToggleButtonGadget	
user_defined	

XmSpinBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNactivateCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmodifyVerifyCallback

XmSpinBox	
Controls	**Reasons**
XmCascadeButtonGadget	XmNpopupHandlerCallback
XmCheckBox	XmNvalueChangedCallback
XmComboBox	XmNlosingFocusCallback
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	

XmSpinBox	
Controls	**Reasons**
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	

XmSpinBox	
Controls	**Reasons**
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmTab	
Controls	**Reasons**
No children are supported	MrmNcreateCallback
	XmNdestroyCallback

XmTabList	
Controls	**Reasons**
XmTab	MrmNcreateCallback
	XmNdestroyCallback

XmTearOffButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNconvertCallback
	XmNdestroyCallback
	XmNdisarmCallback

XmTearOffButton	
Controls	**Reasons**
	XmNhelpCallback
	XmNpopupHandlerCallback

XmTemplateDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback
XmCheckBox	XmNmapCallback
XmComboBox	XmNokCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusMovedCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	

XmTemplateDialog	
Controls	**Reasons**
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	

XmTemplateDialog	
Controls	**Reasons**
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmText	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
XmRenderTable	XmNactivateCallback
	XmNconvertCallback
	XmNdestinationCallback
	XmNdestroyCallback
	XmNfocusCallback
	XmNgainPrimaryCallback
	XmNhelpCallback

XmText	
Controls	**Reasons**
	XmNlosePrimaryCallback
	XmNlosingFocusCallback
	XmNmodifyVerifyCallback
	XmNmodifyVerifyCallbackWcs
	XmNmotionVerifyCallback
	XmNpopupHandlerCallback
	XmNvalueChangedCallback

XmTextField	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
XmRenderTable	XmNactivateCallback
	XmNconvertCallback
	XmNdestinationCallback
	XmNdestroyCallback
	XmNfocusCallback
	XmNgainPrimaryCallback
	XmNhelpCallback
	XmNlosePrimaryCallback
	XmNlosingFocusCallback
	XmNmodifyVerifyCallback
	XmNmodifyVerifyCallbackWcs
	XmNmotionVerifyCallback
	XmNpopupHandlerCallback
	XmNvalueChangedCallback

XmToggleButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNarmCallback
	XmNconvertCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback
	XmNpopupHandlerCallback
	XmNvalueChangedCallback

XmToggleButtonGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback
	XmNvalueChangedCallback

XmWarningDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNlosingFocusCallback

XmWarningDialog	
Controls	**Reasons**
XmCheckBox	XmNmapCallback
XmComboBox	XmNokCallback
XmCommand	XmNpopdownCallback
XmCommandDialog	XmNpopupCallback
XmContainer	XmNpopupHandlerCallback
XmDialogShell	XmNrealizeCallback
XmDrawingArea	XmNunmapCallback
XmDrawnButton	XmNfocusMovedCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	

XmWarningDialog	
Controls	**Reasons**
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	

XmWarningDialog	
Controls	**Reasons**
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmWorkArea	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNpopupHandlerCallback
XmCascadeButtonGadget	XmNtearOffMenuActivateCallback
XmCheckBox	XmNtearOffMenuDeactivateCallback
XmComboBox	XmNunmapCallback
XmCommand	XmNmapCallback
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	

XmWorkArea	
Controls	**Reasons**
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	

XmWorkArea	
Controls	**Reasons**
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmWorkingDialog	
Controls	**Reasons**
XmArrowButton	
XmArrowButtonGadget	
XmBulletinBoard	
XmBulletinBoardDialog	
XmCascadeButton	

XmWorkingDialog	
Controls	**Reasons**
XmCascadeButtonGadget	
XmCheckBox	
XmComboBox	
XmCommand	
XmCommandDialog	
XmContainer	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmIconGadget	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	

XmWorkingDialog	
Controls	**Reasons**
XmNotebook	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRenderTable	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmSpinBox	
XmTemplateDialog	
XmText	
XmTextField	

XmWorkingDialog	
Controls	**Reasons**
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

Appendix C

UIL Arguments

This appendix provides an alphabetical listing of the UIL arguments and their data types. Each argument name is the same as the corresponding Motif Toolkit resource name. For information on which arguments are supported for which objects and for the default values of arguments, see the widget reference pages.

UIL Argument Name	Argument Type
XmNaccelerator	string
XmNacceleratorText	compound_string
XmNaccelerators	translation_table
XmNadjustLast	boolean
XmNadjustMargin	boolean
XmNalignment	unsigned char
XmNallowOverlap	boolean
XmNallowResize	boolean
XmNallowShellResize	boolean

UIL Argument Name	Argument Type
XmNancestorSensitive	boolean
XmNapplyLabelString	compound_string
XmNarmColor	color
XmNarmPixmap	pixmap
XmNarrowDirection	integer
XmNarrowLayout	unsigned char
XmNarrowOrientation	unsigned char
XmNarrowSensitivity	integer
XmNarrowSize	horizontal_float
XmNarrowSpacing	horizontal_float
XmNaudibleWarning	integer
XmNautoDragModel	integer
XmNautoShowCursorPosition	boolean
XmNautoUnmanage	boolean
XmNautomaticSelection	boolean
XmNbackPageBackground	color
XmNbackPageForeground	color
XmNbackPageNumber	integer
XmNbackPagePlacement	integer
XmNbackPageSize	horizontal_float
XmNbackground	color
XmNbackgroundPixmap	pixmap
XmNbaseHeight	vertical_float
XmNbaseWidth	horizontal_float
XmNbindingPixmap	pixmap
XmNbindingType	integer

UIL Argument Name	Argument Type
XmNbindingWidth	horizontal_float
XmNblinkRate	integer
XmNborderColor	color
XmNborderPixmap	pixmap
XmNborderWidth	horizontal_float
XmNbottomAttachment	integer
XmNbottomOffset	vertical_float
XmNbottomPosition	integer
XmNbottomShadowColor	color
XmNbottomShadowPixmap	pixmap
XmNbottomWidget	widget_ref
XmNbuttonCount	integer
XmNbuttonFontList	font_table
XmNbuttonRenderTable	widget_ref
XmNbuttons	string_table
XmNcancelButton	widget_ref
XmNcancelLabelString	compound_string
XmNcascadePixmap	pixmap
XmNchildHorizontalAlignment	integer
XmNchildHorizontalSpacing	horizontal_float
XmNchildPlacement	integer
XmNchildType	integer
XmNchildVerticalAlignment	integer
XmNcollapsedStatePixmap	pixmap
XmNcolormap	identifier
XmNcolumns	short

UIL Argument Name	Argument Type
XmNcomboBoxType	integer
XmNcommand	compound_string
XmNcommandWindow	widget_ref
XmNcommandWindowLocation	integer
XmNcreatePopupChildProc	any
XmNcurrentPageNumber	integer
XmNcursorPosition	integer
XmNcursorPositionVisible	boolean
XmNdarkThreshold	integer
XmNdecimal	string
XmNdecimalPoints	integer
XmNdefaultArrowSensitivity	integer
XmNdefaultButton	widget_ref
XmNdefaultButtonShadowThickness	horizontal_float
XmNdefaultButtonType	integer
XmNdefaultFontList	font_table
XmNdefaultPixmapResolution	unsigned short
XmNdefaultPosition	boolean
XmNdeleteResponse	integer
XmNdepth	identifier
XmNdetail	string_table
XmNdetailColumnHeading	string_table
XmNdetailColumnHeadingCount	integer
XmNdetailCount	integer
XmNdetailOrder	integer_table
XmNdetailOrderCount	integer

UIL Argument Name	Argument Type
XmNdetailTabList	widget_ref
XmNdialogStyle	integer
XmNdialogTitle	compound_string
XmNdialogType	integer
XmNdirListItemCount	integer
XmNdirListItems	string_table
XmNdirListLabelString	compound_string
XmNdirMask	compound_string
XmNdirSearchProc	any
XmNdirSpec	compound_string
XmNdirectory	compound_string
XmNdirectoryValid	boolean
XmNdoubleClickInterval	integer
XmNeditMode	integer
XmNeditable	boolean
XmNeditingPath	integer
XmNenableThinThickness	boolean
XmNendJobCallback	XtCallbackList
XmNentryAlignment	integer
XmNentryBorder	horizontal_float
XmNentryClass	class_rec_name
XmNentryParent	widget_ref
XmNentryVerticalAlignment	integer
XmNentryViewType	integer
XmNexpandedStatePixmap	pixmap
XmNfileListItemCount	integer

UIL Argument Name	Argument Type
XmNfileListItems	XmStringTable
XmNfileListLabelString	compound_string
XmNfileSearchProc	any
XmNfileTypeMask	integer
XmNfillOnArm	boolean
XmNfillOnSelect	boolean
XmNfilterLabelString	compound_string
XmNfirstPageNumber	integer
XmNfont	font
XmNfontList	font_table
XmNfontName	string
XmNfontType	integer
XmNforeground	color
XmNforegroundThreshold	integer
XmNfractionBase	integer
XmNframeBackground	color
XmNframeChildType	integer
XmNframeShadowThickness	horizontal_float
XmNgeometry	string
XmNheight	vertical_float
XmNheightInc	vertical_float
XmNhelpLabelString	compound_string
XmNhighlightColor	color
XmNhighlightOnEnter	boolean
XmNhighlightPixmap	pixmap
XmNhighlightThickness	horizontal_float

UIL Argument Name	Argument Type
XmNhistoryItemCount	integer
XmNhistoryItems	string_table
XmNhistoryMaxItems	integer
XmNhistoryVisibleItemCount	integer
XmNhorizontalScrollBar	widget_ref
XmNhorizontalSpacing	horizontal_float
XmNiconMask	pixmap
XmNiconPixmap	pixmap
XmNiconWindow	any
XmNiconX	horizontal_float
XmNiconY	vertical_float
XmNincrement	integer
XmNincrementValue	integer
XmNindeterminatePixmap	pixmap
XmNindicatorOn	integer
XmNindicatorSize	horizontal_float
XmNindicatorType	integer
XmNinitialDelay	integer
XmNinitialFocus	widget_ref
XmNinitialResourcesPersistent	boolean
XmNinitialState	integer
XmNinput	boolean
XmNinputMethod	string
XmNinputPolicy	integer
XmNinsertPosition	identifier
XmNisAligned	boolean

UIL Argument Name	Argument Type
XmNisHomogeneous	boolean
XmNitemCount	integer
XmNitems	string_table
XmNkeyboardFocusPolicy	integer
XmNlabelFontList	font_table
XmNlabelInsensitivePixmap	pixmap
XmNlabelPixmap	pixmap
XmNlabelRenderTable	widget_ref
XmNlabelString	compound_string
XmNlabelType	integer
XmNlargeCellHeight	vertical_float
XmNlargeCellWidth	horizontal_float
XmNlargeIconMask	pixmap
XmNlargeIconPixmap	pixmap
XmNlargeIconX	horizontal_float
XmNlargeIconY	vertical_float
XmNlastPageNumber	integer
XmNlayoutDirection	integer
XmNlayoutType	integer
XmNleftAttachment	integer
XmNleftOffset	horizontal_float
XmNleftPosition	integer
XmNleftWidget	widget_ref
XmNlightThreshold	integer
XmNlist	widget
XmNlistItemCount	integer

UIL Argument Name	Argument Type
XmNlistItems	string_table
XmNlistLabelString	compound_string
XmNlistMarginHeight	vertical_float
XmNlistMarginWidth	horizontal_float
XmNlistSizePolicy	integer
XmNlistSpacing	horizontal_float
XmNlistUpdated	boolean
XmNlistVisibleItemCount	integer
XmNloadModel	integer
XmNmainWindowMarginHeight	vertical_float
XmNmainWindowMarginWidth	horizontal_float
XmNmajorTabSpacing	vertical_float
XmNmappedWhenManaged	boolean
XmNmappingDelay	integer
XmNmargin	horizontal_float
XmNmarginBottom	vertical_float
XmNmarginHeight	dimension
XmNmarginLeft	horizontal_float
XmNmarginRight	horizontal_float
XmNmarginTop	vertical_float
XmNmarginWidth	dimension
XmNmatchBehavior	integer
XmNmaxAspectX	integer
XmNmaxHeight	vertical_float
XmNmaxLength	integer
XmNmaxWidth	horizontal_float

UIL Argument Name	Argument Type
XmNmaximum	integer
XmNmaximumValue	integer
XmNmenuAccelerator	string
XmNmenuBar	widget_ref
XmNmenuHelpWidget	widget_ref
XmNmenuHistory	widget_ref
XmNmenuPost	string
XmNmessageAlignment	integer
XmNmessageString	compound_string
XmNmessageWindow	widget_ref
XmNminAspectX	integer
XmNminAspectY	integer
XmNminHeight	vertical_float
XmNminWidth	horizontal_float
XmNminimizeButtons	boolean
XmNminimum	integer
XmNminimumValue	integer
XmNminorTabSpacing	vertical_float
XmNmnemonic	keysym
XmNmnemonic	keysym
XmNmnemonicCharSet	string
XmNmultiClick	integer
XmNmustMatch	boolean
XmNmwmDecorations	integer
XmNmwmFunctions	integer
XmNmwmInputMode	integer

UIL Argument Name	Argument Type
XmNmwmMenu	string
XmNnavigationType	integer
XmNnavigationType	integer
XmNnoMatchString	compound_string
XmNnoResize	boolean
XmNnotebookChildType	integer
XmNnumColumns	integer
XmNnumValues	integer
XmNoffsetModel	integer
XmNokLabelString	compound_string
XmNorientation	integer
XmNoutlineButtonPolicy	integer
XmNoutlineColumnWidth	horizontal_float
XmNoutlineIndentation	horizontal_float
XmNoutlineLineStyle	integer
XmNoutlineState	integer
XmNoverrideRedirect	boolean
XmNpacking	integer
XmNpageIncrement	integer
XmNpageNumber	integer
XmNpageSetupCallback	XtCallbackList
XmNpaneMaximum	horizontal_float
XmNpattern	compound_string
XmNpdmNotificationCallback	XtCallbackList
XmNpendingDelete	boolean
XmNpopupEnabled	boolean

UIL Argument Name	Argument Type
XmNposition	integer
XmNpositionIndex	integer
XmNpositionMode	XtEnum
XmNpositionType	unsigned char
XmNpreeditType	string
XmNprimaryOwnership	integer
XmNprintOrientation	string
XmNprintOrientations	string
XmNprintResolution	unsigned int
XmNprintResolutions	unsigned int*
XmNprocessingDirection	integer
XmNpromptString	compound_string
XmNpushButtonEnabled	boolean
XmNqualifySearchDataProc	any
XmNradioAlwaysOne	boolean
XmNradioBehavior	boolean
XmNrecomputeSize	boolean
XmNrefigureMode	boolean
XmNrenderTable	widget_ref
XmNrepeatDelay	integer
XmNresizable	boolean
XmNresizeHeight	boolean
XmNresizePolicy	integer
XmNresizeWidth	boolean
XmNrightAttachment	integer
XmNrightOffset	horizontal_float

UIL Argument Name	Argument Type
XmNrightPosition	integer
XmNrightWidget	widget_ref
XmNrowColumnType	integer
XmNrows	short
XmNrubberPositioning	boolean
XmNsashHeight	horizontal_float
XmNsashIndent	horizontal_float
XmNsashShadowThickness	horizontal_float
XmNsashWidth	horizontal_float
XmNsaveUnder	boolean
XmNscaleHeight	vertical_float
XmNscaleMultiple	integer
XmNscaleWidth	horizontal_float
XmNscreen	identifier
XmNscrollBarDisplayPolicy	integer
XmNscrollBarPlacement	integer
XmNscrollHorizontal	boolean
XmNscrollLeftSide	boolean
XmNscrollTopSide	boolean
XmNscrollVertical	boolean
XmNscrolledWindowChildType	integer
XmNscrolledWindowMarginHeight	integer
XmNscrolledWindowMarginWidth	integer
XmNscrollingPolicy	integer
XmNselectColor	color
XmNselectInsensitivePixmap	pixmap

UIL Argument Name	Argument Type
XmNselectPixmap	pixmap
XmNselectThreshold	integer
XmNselectedItem	compound_string
XmNselectedItemCount	integer
XmNselectedItems	string_table
XmNselectedPosition	integer
XmNselectedPositionCount	integer
XmNselectedPositions	integer_table
XmNselectionArray	integer_table
XmNselectionArrayCount	integer
XmNselectionLabelString	compound_string
XmNselectionMode	integer
XmNselectionPolicy	integer
XmNselectionTechnique	integer
XmNsensitive	boolean
XmNseparatorOn	boolean
XmNseparatorType	integer
XmNset	integer
XmNshadowThickness	horizontal_float
XmNshadowType	integer
XmNshellUnitType	integer
XmNshowArrows	integer
XmNshowAsDefault	integer
XmNshowSeparator	boolean
XmNshowValue	integer
XmNskipAdjust	boolean

UIL Argument Name	Argument Type
XmNsliderSize	integer
XmNsliderVisual	integer
XmNslidingMode	integer
XmNsmallCellHeight	vertical_float
XmNsmallCellWidth	horizontal_float
XmNsmallIconMask	pixmap
XmNsmallIconPixmap	pixmap
XmNsmallIconX	horizontal_float
XmNsmallIconY	vertical_float
XmNsnapBackMultiple	integer
XmNsource	any
XmNspacing	dimension
XmNspatialIncludeModel	integer
XmNspatialResizeModel	integer
XmNspatialSnapModel	integer
XmNspatialStyle	integer
XmNspinBoxChildType	integer
XmNstartJobCallback	XtCallbackList
XmNstrikethruType	integer
XmNstringDirection	integer
XmNsubMenuId	widget_ref
XmNsymbolPixmap	pixmap
XmNtabValue	float
XmNtag	string
XmNtearOffModel	integer
XmNtearOffTitle	string

UIL Argument Name	Argument Type
XmNtextAccelerators	translation_table
XmNtextColumns	integer
XmNtextField	widget
XmNtextFontList	font_table
XmNtextPath	integer
XmNtextRenderTable	widget_ref
XmNtextString	compound_string
XmNtextTranslations	translation_table
XmNtitle	string
XmNtitleEncoding	any
XmNtitleString	compound_string
XmNtoggleMode	integer
XmNtopAttachment	integer
XmNtopCharacter	integer
XmNtopItemPosition	integer
XmNtopOffset	vertical_float
XmNtopPosition	integer
XmNtopShadowColor	color
XmNtopShadowPixmap	pixmap
XmNtopWidget	widget_ref
XmNtotalLines	integer
XmNtransient	boolean
XmNtransientFor	widget_ref
XmNtranslations	translation_table
XmNtraversalOn	boolean
XmNtroughColor	color

UIL Argument Name	Argument Type
XmNunderlineType	integer
XmNunitType	integer
XmNunselectColor	color
XmNuseAsyncGeometry	boolean
XmNuserData	any
XmNvalue	any
XmNvalueWcs	wide_character
XmNvalues	string_table
XmNverifyBell	boolean
XmNverifyPreedit	boolean
XmNverticalScrollBar	widget_ref
XmNverticalSpacing	vertical_float
XmNviewType	integer
XmNvisibleItemCount	integer
XmNvisibleWhenOff	boolean
XmNvisual	any
XmNvisualEmphasis	integer
XmNvisualPolicy	integer
XmNwaitForWm	boolean
XmNwhichButton	integer
XmNwidth	horizontal_float
XmNwidthInc	horizontal_float
XmNwinGravity	integer
XmNwindowGroup	any
XmNwmTimeout	integer
XmNwordWrap	boolean

UIL Argument Name	Argument Type
XmNworkWindow	widget_ref
XmNx	horizontal_float
XmNy	vertical_float

Appendix D

UIL Diagnostic Messages

This appendix lists the diagnostic messages produced by the UIL compiler. The severity, a description of the message, and a suggestion for correcting the problem are listed for each message. The following strings are used to represent data that varies in the actual message you receive from the UIL compiler:

String	Data Represented
%c	Character
%d	Decimal number
%s	String

Messages are listed alphabetically by IDENT code.

add_source additional UIL source file: %s was ignored

Severity: Error More than one source file was specified. Only the first source file will be compiled.

User Action: Compile additional source files using separate invocations of the compiler.

arg_count procedure %s was previously declared with %d arguments

Severity: Error The declaration of the marked procedure specified a different number of arguments than are present in this procedure reference.

User Action: Check that you are calling the correct function. If you intend to call the procedure with a varying number of arguments, omit the argument list in the procedure declaration.

arg_type found %s value - procedure %s argument must be %s value

Severity: Error The declaration of the marked procedure specified a different type of argument than is present in this procedure reference.

User Action: Check that you are passing the correct argument to the correct function. If you intend to call the procedure with varying argument types, declare the procedure specifying *any* for the type of the argument.

backslash_ignored
unknown escape sequence "\%c" - ignored

Severity: Error A backslash was followed by an unknown escape character. The \ (backslash) is the escape character in UIL. A selected set of single characters can follow a backslash such as \n for newline or \\ to insert a backslash. The character following the backslash was not one of the selected set.

User Action: If you want to add a backslash, use \\. See the **UIL(5X)** reference page for a description of the supported escape sequences.

bad_database
error reading binary database

Severity: Severe The compiler encountered an error in reading a binary widget meta-language description file.

User Action: Check that the file specified to the **--wmd** command line argument is a valid widget meta-language description file.

bad_lang_value
$LANG contains an unknown character set

Severity: Error The character set portion of the locale specified in the LANG environment variable does not correspond to one of the character sets known to the UIL compiler.

User Action: See the **UIL(5X)** reference page for a description of the supported character sets. Change the value of $LANG to contain one of the known character sets.

bug_check internal error: %s

Severity: Severe The compiler diagnosed an internal error.

User Action: Submit a software problem report.

cannot_convert
cannot convert %s type to %s type

Severity: Error The compiler could not perform the specified implicit type conversion.

User Action: Check that the value being specified is what is desired and that the type of the value being specified can be converted to the requested type. See the **UIL(5X)** reference page for a description of standard type conversions and Appendix C for a list of UIL argument types.

circular_def widget %s is part of a circular definition

Severity: Error The indicated item contains a reference to the widget within which it is defined, either within its own definition or within the definition of one of the objects in the widget tree it controls.

User Action: Change the definition of the indicated item so that it does not reference the widget within which it is defined.

circular_ref the %s value is circularly defined

Severity: Error The indicated value is referenced either within its own declaration or recursively within the declaration of one of the values it depends on.

User Action: Change the declaration of the indicated value so that it does not depend on itself.

control_char
unprintable character \%d\ ignored

Severity: Error The compiler encountered an illegal control character in the UIL specification file. The decimal value of the character is given between the \ (backslash) characters.

User Action: Replace the character with the sequence specified in the message (for example, \3 if the control character's internal value is 3). UIL provides several built-in control characters such as \n and \r for newline and carriage return. See the **UIL(5X)** reference page for a complete list of supported escape sequences.

create_proc

creation procedure is not supported by the %s widget

Severity: Error You specified a creation procedure for a Motif Toolkit widget. You can specify a creation procedure only for a user-defined widget.

User Action: Remove the procedure clause following the object type.

create_proc_inv

creation procedure is not allowed in a %s widget reference

Severity: Error You specified a creation procedure when referencing an object. You can specify a creation procedure only when you declare the object.

User Action: Remove the procedure clause following the object type.

create_proc_req

creation procedure is required in a %s widget declaration

Severity: Error When defining a user-defined widget, you must specify the name of the creation function for creating an instance of this widget.

User Action: Insert a procedure clause following the widget type in the widget declaration. You also need to declare the creation procedure using a procedure declaration. For example:

```
procedure my_creation_proc();
object list_box:
  user_defined procedure
      my_creation_proc()
      { arguments ... };
```

ctx_req context requires a %s - %s was specified

Severity: Error At the point marked in the specification, one type of object (such as a widget) is required and your specification supplied a different type of object (such as value).

User Action: Check for misspelling or that you have referred to the intended object.

default_charset

%s used as charset name; %s used as charset component

Severity: Informational If UIL encounters a character set that is neither a built-in character set nor user-defined, the character set of the string will be set to **XmFONTLIST_DEFAULT_TAG**. This message is printed for each unique chararacter that it had to set. In the message, the first string is the character set name the user used, and the second string is usually **XmFONTLIST_DEFAULT_TAG**.

different_units

incompatible unit types for arithmetic operation

Severity: Severe Check for incompatible unit types for the requested arithmetic operation.

dup_letter color letter used for prior color in this table

Severity: Error Each of the letters used to represent a color in a color table must be unique. If not, that letter in an icon would represent more than one color; each pixel can have only one color associated with it at a time. The letter marked has been assigned to more than one color.

User Action: Choose which color the letter is to represent and remove any duplicates or assign them a new character.

dup_list %s %s already specified for this %s %s

Severity: Error A widget or gadget declaration can have at most one arguments list, one callbacks list, and one controls list.

User Action: If you want to specify multiple lists of arguments, controls, and callbacks, you can do so within one list. For example:

```
arguments { arguments_list1;
   arguments_list2; };
```

dupl_opt duplicate option \ "%s" \ was ignored

Severity: Warning The same command line option has been repeated more than once (for example, the "-o" option or the "-v" option)

User Action: Remove duplicate command line option.

future_version
binary database compiled with a future version

Severity: Severe The binary widget meta-language description file was compiled with a later version of Motif than that used by the UIL compiler.

User Action: Either get a later version of the UIL compiler or a widget meta-language description file compiled with an earlier version of Motif.

gadget_not_sup
%s gadget is not supported - %s widget will be used instead

Severity: Warning The indicated object type does not support a gadget variant; only a widget variant is supported for this object type. The UIL compiler ignores the gadget indication, and creates widgets of this object type.

User Action: Specify that this object type is a widget instead of a gadget.

icon_letter row %d, column %d: letter \"%c"\ not in color table

Severity: Error You have specified a color to be used in an icon that is not in that icon's color table. The invalid color is identified in the message by displaying the letter used to represent that color between the \ (backslashes). This letter was not defined in the specified color table.

User Action: Either add the color to the icon's color table or use a character representing a color in the color table. The default color table defines ''(space) as background and '*' (asterisk) as foreground.

icon_width row %d must have same width as row 1

Severity: Error The icons supported by UIL are rectangular (that is, x pixels wide by y pixels high). As a result, each of the strings used to represent a row of pixels in an icon must have the same length. The specified row does not have the same length as the first row.

User Action: Make all the strings in the icon function the same length.

include_file invalid include file name

Severity: Severe The include file name was not specified as a string literal.

User Action: Ensure that the include file name is a single or double quoted string literal and not a concatenated string or a value reference.

inv_module invalid module structure - check UIL module syntax

Severity: Error The structure of the UIL module is incorrect.

User Action: If there are any syntax errors reported, fix them and recompile. For example, if the error occurs before the first object declaration (that is, before your value and object declarations), check the syntax of the module header for unwanted; (semicolons) after the module clauses. If the error occurs at the end of the module, check that the module concludes with the keywords "end module;".

invalid_enumval
the %s argument does not support the %s enumerated value

Severity: Warning The indicated enumerated value is not valid for the indicated argument.

User Action: Check the documentation of the indicated argument to determine the correct enumerated values it supports.

list_item %s item not allowed in %s %s

Severity: Error The indicated list item is not of the type required by the list. Arguments lists must contain argument entries, callbacks lists must contain callback entries, controls lists must contain control entries, and procedures lists must contain callback entries.

User Action: Check the syntax for the type of list entry that is required in this context and change the indicated list item.

listing_open
error opening listing file: %s

Severity: Severe The compiler could not create the listing file noted in the message.

User Action: Check that you have write access to the directory you specified to hold the listing file.

listing_write
error writing to listing file: %s

Severity: Severe The compiler could not write a line into the listing file noted in the message.

User Action: Check to see that there is adequate space on the disk specified to hold the listing file.

miss_opt_arg
%s missing following \ "%s" \ option

Severity: Error You used a command line option that requires an argument and you did not provide that argument.

User Action: Omit the option or provide the argument.

name_too_long
name exceeds 31 characters - truncated to: %s

Severity: Error The UIL compiler encountered a name longer than 31 characters. The compiler truncated the name to the leftmost 31 characters.

User Action: Shorten the name in the UIL module source.

names place names clause before other module clauses

Severity: Error The case-sensitivity clause, if specified, must be the first clause following the module's name. You have inserted another module clause before this clause.

User Action: Reorder the module clauses so that the case-sensitivity clause is first.

never_def %s %s was never defined

Severity: Error Certain UIL objects such as gadgets and widgets can be referred to before they are defined. The marked object is such an object. However, the compiler never found the object's declaration.

User Action: Check for misspelling. If the module is case sensitive, the spellings of names in declarations and in references must match exactly.

no_enumset the %s argument does not support enumerated values

Severity: Warning The indicated argument does not support enumerated values.

User Action: Check the documentation of the argument to determine the correct type of value to provide for it.

no_source no source file specified

Severity: Severe No source file was specified to compile.

User Action: Specify the name of a UIL specification file to compile.

no_uid no UID file was produced

Severity: Informational If the compiler reported error or severe diagnostics (that is, any of the diagnostic abbreviations starting with %UIL-E or %UIL-F), a UID file is not created. This diagnostic informs you that the compiler did not produce a UID file.

User Action: Fix the problems reported by the compiler.

non_pvt value used in this context must be private

Severity: Error A private value is one that is not imported or exported. In the context marked by the message, only a private value is legal. Situations where this message is issued include defining one value in terms of another, and arguments to functions. In general, a value must be private when the compiler must know the value at compilation time. Exported values are disallowed in these contexts, even though a value is present, because that value could be overridden at run time.

User Action: Change the value to be private.

not_impl %s is not implemented yet

Severity: Error You are using a feature of UIL that has not been implemented.

User Action: Try an alternate technique.

null a NULL character in a string is not supported

Severity: Warning You have created a string that has an embedded null character. Strings are represented in a UID file and in many Motif Toolkit data structures as null terminated strings. So, although the embedded nulls will be placed in the UID file, Motif Toolkit functions may interpret an embedded null as the terminator for the string.

User Action: Be very careful using embedded nulls.

obj_type found %s %s when expecting %s %s

Severity: Error Most arguments take values of a specific type. The value specified is not correct for this argument.

User Action: The message indicates the expected type of argument. Check that you have specified the intended value and that you specified the correct argument.

operand_type
%s type is not valid for %s

Severity: Error The indicated operand is not of a type that is supported by this operator.

User Action: Check the definition of the operator and make sure the type of the operand you specify is supported by the operator.

out_of_memory
compiler ran out of virtual memory

Severity: Severe The compiler ran out of virtual memory.

User Action: Reduce the size of your application.

out_range value of %s is out of range %s

Severity: Error The value specified is outside the legal range of its type.

User Action: Change the UIL module source.

override_builtin
overriding built-in name %s

Severity: Warning The name marked by the message is the same as the name of a built-in UIL name such as an argument name.

User Action: Be certain that you really want to override the particular name. If not, change the name being declared.

prev_error compilation terminated - fix previous errors

Severity: Severe Errors encountered during the compilation have caused the compiler to abort.

User Action: Fix the errors already diagnosed by the compiler and recompile.

previous_def
name %s previously defined as %s

Severity: Error The name marked by the message was used in a previous declaration. UIL requires that the names of all objects declared within a module be unique.

User Action: Check for a misspelling. If the module is case sensitive, the spellings of names in declarations and in references must match exactly.

single_control
%s widget supports only a single control

Severity: Warning The indicated widget maps a single control to a subtree resource. For example, the pulldown control of a cascade button is mapped to the **XmNsubMenuId** resource. However, you have specified more than one control, resulting in the message.

User Action: Remove the extra children in the controls list.

single_letter color letter string must be a single character

Severity: Error The string associated with each color in a color table must hold exactly one character. You have specified a string with either fewer or more characters.

User Action: Use a single character to represent each color in a color table.

single_occur
%s %s supports only a single %s %s

Severity: Warning You have specified a particular clause more than once in a context where that clause can occur only once. For example, the version clause in the module can only occur once.

User Action: Choose the correct clause and delete the others.

src_close error closing source file: %s

Severity: Warning Some error occurred while closing the indicated source file.

User Action: Check for operating or file system conditions that may have caused this problem. If the output file was successfully created, this warning can probably be ignored. If the output file was not created, attempt the compilation again.

src_limit too many source files open: %s

Severity: Severe The compiler has a fixed limit of 100 for the number of source and include files that it can process. The file exceeding this limit is reported in the message.

User Action: Use fewer include files.

src_null_char
source line contains a null character

Severity: Error The specified source line contains a null character. The compiler ignores any text following the null character.

User Action: Replace each null character with the escape sequence \ (backslash).

src_open error opening source file: %s

Severity: Severe The compiler could not open the UIL specification file listed in the message.

User Action: Check that the file listed in the message is the one you want to compile, that it exists, and that you have read access to the file. If you are using a large number of include files, you may have exceeded your quota for open files.

src_read error reading next line of source file: %s

Severity: Severe The compiler could not read a line of the UIL specification file listed in the message.

User Action: In the listing file, this message should appear following the last line the compiler read successfully. First check that the file you are compiling is a UIL specification file. If it is, the file most likely contains corrupted records.

src_truncate
line truncated at %d characters

Severity: Error The compiler encountered a source line greater than 132 characters. Characters beyond the 132 character limit were ignored.

User Action: Break each source line longer than 132 characters into several source lines. Long string literals can be created using the concatenation operator.

submit_spr internal error - submit defect report

Severity: Severe The compiler diagnosed an internal error.

User Action: Get a listing and look where the error is being issued. Try fixing any faulty syntax in this area. If you are unable to prevent this error, submit a software problem report.

summary errors: %d warnings: %d informationals: %d

Severity: Informational This message lists a summary of the diagnostics issued by the compiler, and appears only when diagnostics have been issued.

User Action: Fix the problems reported. You can use the **--w** option qualifier to suppress informational and warning diagnostics that you have determined to be harmless.

supersede this %s %s supersedes a previous definition in this %s %s

Severity: Informational An argument or callback list has either a duplicate argument or duplicate reason.

User Action: This is not necessarily an error. The compiler is alerting you to make sure that you intended to override a prior argument's value. This informational message can be suppressed using the **--w** option qualifier.

syntax unexpected %s token seen - parsing resumes after \"%c"\

Severity: Error At the point marked in the module, the compiler found a construct such as a punctuation mark, name, or keyword when it was expecting a different construct. The compiler continued analyzing the module at the next occurrence of the construct stated in the message.

User Action: Check the syntax of your UIL module at the point marked by the compiler. If the module specifies case-sensitive names, check that your keywords are in lowercase characters.

too_many too many %ss in %s, limit is %d

Severity: Error You exceeded a compiler limit such as the number of fonts in a font table or the number of strings in a translation table. The message indicates the limit imposed by the compiler.

User Action: Restructure your UIL module.

too_many_dirs
too many \"%s\" options, limit is %d

Severity: Warning You specified too many include directories using the **--I** option. The message indicates the limit imposed by the compiler.

User Action: Reduce the number of include directories specified. If necessary, consolidate some of your include files into fewer directories.

uid_open error opening UID file: %s

Severity: Severe The compiler could not create the UID file noted in the message. A UID file holds the compiled user-interface specification.

User Action: Check that you have write access to the directory you specified to hold the UID file. If you have a large number of source and include files, check that you have not exceeded your open file quota.

uid_write error writing UID file: %s

Severity: Severe An error occurred when trying to write to the indicated UID file.

User Action: Check that you have write access to the directory you specified to hold the UID file and that you have write access to the file, if it already exists. Also check that you have not run out of space in your file system.

undefined %s %s must be defined before this reference

Severity: Error The object pointed to in the message was either never defined or not defined prior to this point in the module. The compiler requires the object to be defined before you refer to the object.

User Action: Check for a misspelling of the object's name, a missing declaration for the object, or declaring the object after its first reference. If names in the module are case sensitive, the spellings of the name in the declaration and in the reference must match exactly.

unknown_charset
unknown character set

Severity: Error The message is pointing to a context where a character set name is required. You have not specified the name of a character set in that context.

User Action: Check for misspelling. A list of the supported character sets is given in the **UIL(5X)** reference page. If you specified case-

sensitive names in the module, check that the character set name is in lowercase characters.

unknown_opt

unknown option \ "%s" \ was ignored

Severity: Warning An unknown option has been used in the compiler command line.

User Action: Check what you typed on the command line.

unknown_seq

unknown sequence \"%s"\ ignored

Severity: Error The compiler detected a sequence of printable characters it did not understand. The compiler omitted the sequence of characters listed between the " " (double quotation marks).

User Action: Fix the UIL module source.

unsupp_const

the %s constraint is not supported for the %s %s

Severity: Warning The particular constraint argument you specified is not supported for the indicated widget or gadget parent.

User Action: See the UIL built-in tables in Appendix A for the constraints supported for the children of each object. If a widget creation function accepts a constraint argument that UIL rejects, this does not necessarily indicate that the UIL compiler is in error. Widget creation functions ignore arguments that they do not support, without notifying you that the argument is being ignored.

unsupported

the %s %s is not supported for the %s object

Severity: Warning Each widget or gadget supports a specific set of arguments, reasons, and children. The particular argument, reason, or child you specified is not supported for this widget or gadget.

User Action: See the UIL built-in tables in Appendix A and Appendix B for the arguments, reasons, and children supported for each object. If a widget creation function accepts an argument that UIL rejects, this does not necessarily indicate that the UIL compiler is in error. Widget creation functions ignore arguments that they do not support, without notifying you that the argument is being ignored.

unterm_seq %s not terminated %s

Severity: Error The compiler detected a sequence that was not properly terminated, such as a string literal without the closing quotation mark.

User Action: Insert the proper termination characters.

value_too_large
value %s is too large for context buffer

Severity: Severe The compiler could not allocate enough memory to temporarily store the indicated value.

User Action: Reduce the size of the object being assigned to the indicated value.

widget_cycle
the %s object's controls hierarchy contains a reference to itself

Severity: Error The indicated object is referenced as a descendant of itself, either within its own definition or within the definition of one of the objects in the widget tree it controls.

User Action: Change the definition of the indicated object so that it is not a descendant of itself.

wmd_open error opening database file: %s

Severity: Severe The compiler could not open the widget meta-language description file listed in the message.

User Action: Check that the file listed in the message is the one you want to use, that it exists, and that you have read access to the file.

wrong_type found %s value when expecting %s value

Severity: Error The indicated value is not of the specific type required by UIL in this context.

User Action: Check the definition of the function or clause.