IF/Prolog V5.2

OSF/Motif Interface

Siemens AG Austria
Is there anything you would like to tell us about this manual?
Please send us your comments.

Siemens AG Austria
PSE KB B2
Gudrunstrasse 11
A-1100 Vienna
Austria
Fax.: +43-1-1707 56992
email: prolog@siemens.at

The information in this document is subject to change and does not represent a commitment on the part of Siemens AG Austria. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

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

X/Open and the X device are trademarks of X/Open Company Ltd.

Copyright ©Siemens AG Austria, 1999. All rights reserved.
The reproduction, transmission, translation or exploitation of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. Delivery subject to availability; right of technical modifications reserved.
## Contents

**Preface**  
1

**Using IF/Prolog for OSF/Motif programming**  
5

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to the X Window System and OSF/Motif</td>
<td>6</td>
</tr>
<tr>
<td>Clients and Server</td>
<td>6</td>
</tr>
<tr>
<td>OSF/Motif and the X Window System</td>
<td>6</td>
</tr>
<tr>
<td>Basic programming concepts for OSF/Motif</td>
<td>7</td>
</tr>
<tr>
<td>OSF/Motif programming using IF/Prolog</td>
<td>12</td>
</tr>
<tr>
<td>Differences between the <em>IF/Prolog OSF/Motif Interface</em> and the OSF/Motif C interface</td>
<td>12</td>
</tr>
<tr>
<td>Compound strings and font lists</td>
<td>14</td>
</tr>
<tr>
<td>Callback structures</td>
<td>14</td>
</tr>
<tr>
<td>Cut and paste functions</td>
<td>15</td>
</tr>
<tr>
<td>Restrictions</td>
<td>16</td>
</tr>
<tr>
<td>Error handling</td>
<td>16</td>
</tr>
<tr>
<td>Terminating IF/Prolog</td>
<td>16</td>
</tr>
<tr>
<td>OSF/Motif data types in IF/Prolog</td>
<td>16</td>
</tr>
<tr>
<td>Structure of an OSF/Motif application</td>
<td>20</td>
</tr>
<tr>
<td>Initializing the X environment</td>
<td>20</td>
</tr>
<tr>
<td>Creating and mapping widgets</td>
<td>20</td>
</tr>
<tr>
<td>Processing user events</td>
<td>21</td>
</tr>
<tr>
<td>Setting and modifying resources</td>
<td>23</td>
</tr>
<tr>
<td>Sample program</td>
<td>24</td>
</tr>
<tr>
<td>User Interface Language UIL</td>
<td>29</td>
</tr>
</tbody>
</table>
3 Widgets and gadgets

- ApplicationShell ..................................................... 37
- ArrowButton (widget) ................................................. 38
- ArrowButton (gadget) .................................................. 40
- BulletinBoard ............................................................. 41
- CascadeButton (widget) .............................................. 45
- CascadeButton (gadget) ............................................. 47
- Command ................................................................. 48
- Composite ................................................................. 50
- Constraint ................................................................. 51
- Core ......................................................................... 52
- DialogShell ................................................................. 55
- Display ...................................................................... 56
- DragContext ............................................................... 58
- DragIcon ..................................................................... 63
- Draw ....................................................................... 65
- DrawingArea ............................................................. 70
- DrawnButton ............................................................. 72
- DropSite ................................................................. 74
- FileSelectionBox ...................................................... 77
- Form ...................................................................... 80
- Frame ................................................................. 85
- Gadget ................................................................. 87
- Label (widget) ........................................................ 90
- Label (gadget) ......................................................... 93
- List ................................................................. 94
- MainWindow .......................................................... 99
- Manager .............................................................. 101
- MenuShell ............................................................ 104
- MessageBox .......................................................... 105
– OverrideShell .................................................. 107
– PanedWindow .................................................. 108
– Primitive ...................................................... 111
– PushButton (widget) ........................................... 114
– PushButton (gadget) .......................................... 116
– RowColumn .................................................... 117
– Scale .......................................................... 126
– Screen ......................................................... 129
– ScrollBar (widget) ............................................ 132
– ScrolledWindow .............................................. 136
– SelectionBox .................................................. 139
– Separator (widget) .......................................... 143
– Separator (gadget) .......................................... 144
– Shell .......................................................... 145
– Text and TextField ......................................... 147
– ToggleButton (widget) ...................................... 154
– ToggleButton (gadget) ...................................... 157
– TopLevelShell ................................................ 158
– TransientShell ............................................... 159
– VendorShell .................................................. 160
– WMShell ...................................................... 164

4 The Draw widget ............................................... 169

5 Overview of predefined predicates by function ........................................... 175

6 Reference section on predefined predicates .......................................... 191

  mrmCloseHierarchy/1 – Close hierarchy of the Motif Resource Manager .......... 194
  mrmFetchColorLiteral/4 – Fetch color value from a UID hierarchy ............ 195
  mrmFetchIconLiteral/6 – Convert icon value from a UID hierarchy into a pixmap .................................................. 196
  mrmFetchLiteral/4 – Fetch value of a UIL variable ............................. 198
mrmFetchSetValues/3 – Set resource values equal ................................ 200
mrmFetchWidget.../4/5 – Create and name a UIL widget ................... 201
mrmInitialize/0 – Initialize Motif Resource Manager (MRM) ............. 203
mrmOpenHierarchy/2 – Create MRM hierarchy id and open associated files 204
mrmRegisterIdentifierNames.../1/2 – Generate assignment of UIL names and values .......................................................... 205
mrmRegisterProcedureNames/1 – Generate assignment of UIL names to call-
back predicates ....................................................................... 206
xmActivate...Protocol/3 – Activate protocols ..................................... 208
xmAddProtocolCallback/5 – Define callback predicate for a protocol .... 209
xmAddProtocols/3 – Define protocols ............................................ 210
xmAddStringToUnitTypeConverter/0 – Specify function for converting char-
acter strings into valid units of measurement .................................. 211
xmAddTabGroup/1 – Add a widget to a tab group ............................... 212
xmAddWMProtocolCallback/4 – Define callback predicate for a WM protocol 213
xmAddWMProtocols/2 – Define WM protocols ................................... 214
xmCascadeButton...Highlight/2 – Draw highlight for the CascadeButton . 215
xmClipboardCancelCopy/2 – Cancel copying of data into the clipboard ... 216
xmClipboardCopy/5 – Start copying data into the clipboard ............... 217
xmClipboardCopyByName/5 – Copy data into the clipboard after a delay .. 219
xmClipboardCopyName/7 – Start delayed copying of data into the clipboard 221
xmClipboardEndCopy/2 – End copying of data into the clipboard .......... 224
xmClipboardEndRetrieve/1 – End copying of data out of the clipboard ... 225
xmClipboardInquireCount/2 – Query the number of formats in the clipboard 226
xmClipboardInquireFormat/3 – Query format name for data in the clipboard 227
xmClipboardInquireLength/3 – Query length of the data in the clipboard . 228
xmClipboardInquirePendingItems/3 – Query pending copy operations .... 229
xmClipboardLock/1 – Lock clipboard ............................................ 230
xmClipboardRegisterFormat/3 – Register the format name for the clipboard 231
xmClipboardRetrieve/4 – Retrieve data from the clipboard ............... 232
xmClipboardStartCopy/3 – Prepare to copy data into the clipboard ....... 234
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmClipboardStartRetrieve/1</code></td>
<td>Start to copy data from the clipboard</td>
</tr>
<tr>
<td><code>xmClipboardUndoCopy/1</code></td>
<td>Delete last data element from the clipboard store</td>
</tr>
<tr>
<td><code>xmClipboardUnlock/2</code></td>
<td>Remove lock from clipboard</td>
</tr>
<tr>
<td><code>xmClipboardWithdrawFormat/2</code></td>
<td>Cancel outstanding data copying operations</td>
</tr>
<tr>
<td><code>xmCommandAppendValue/2</code></td>
<td>Append text to the Command widget command</td>
</tr>
<tr>
<td><code>xmCommandError/2</code></td>
<td>Output error message to the Command widget history area</td>
</tr>
<tr>
<td><code>xmCommandGetChild/3</code></td>
<td>Query a component of the Command widget</td>
</tr>
<tr>
<td><code>xmCommandSetValue/2</code></td>
<td>Overwrite a command in the Command widget command list</td>
</tr>
<tr>
<td><code>xmConvertUnits/6</code></td>
<td>Convert units</td>
</tr>
<tr>
<td><code>xmCreate.../4</code></td>
<td>Create widgets</td>
</tr>
<tr>
<td><code>xmCreateDragIcon/4</code></td>
<td>Create a DragIcon widget</td>
</tr>
<tr>
<td><code>xmCvtCToXmString/2, xmCvtXmStringToCT/2</code></td>
<td>Convert character string</td>
</tr>
<tr>
<td><code>xmDeactivate...Protocol/3</code></td>
<td>Deactivate protocols</td>
</tr>
<tr>
<td><code>xmDragCancel/1</code></td>
<td>Abort drag operation</td>
</tr>
<tr>
<td><code>xmDragStart/3</code></td>
<td>Initiate a drag operation</td>
</tr>
<tr>
<td><code>xmDropSiteConfigureStackingOrder/3</code></td>
<td>Reorder stack of widgets registered as drop sites</td>
</tr>
<tr>
<td><code>xmDropSiteEndUpdate/1</code></td>
<td>Economize multiple calls to <code>xmDropSiteUpdate</code></td>
</tr>
<tr>
<td><code>xmDropSiteQueryStackingOrder/3</code></td>
<td>Retrieve drop site parent and drop site children of widget</td>
</tr>
<tr>
<td><code>xmDropSiteRegister/2</code></td>
<td>Register widget as a drop site</td>
</tr>
<tr>
<td><code>xmDropSiteRetrieve/2</code></td>
<td>Retrieve resource values of a drop site</td>
</tr>
<tr>
<td><code>xmDropSiteStartUpdate/1</code></td>
<td>Economize multiple calls to <code>xmDropSiteUpdate</code></td>
</tr>
<tr>
<td><code>xmDropSiteUnregister/1</code></td>
<td>Free drop site information for widget</td>
</tr>
<tr>
<td><code>xmDropSiteUpdate/2</code></td>
<td>Update resources for registered DropSite</td>
</tr>
<tr>
<td><code>xmDropTransferAdd/2</code></td>
<td>Enable processing of additional entries after initiating a drop transfer</td>
</tr>
<tr>
<td><code>xmDropTransferStart/3</code></td>
<td>Initiate a drop transfer</td>
</tr>
<tr>
<td><code>xmFileSelectionBoxGetChild/3</code></td>
<td>Retrieve components of the FileSelection-Box widget</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xmFileSelectionDoSearch/2</td>
<td>Search file system for a directory or file</td>
</tr>
<tr>
<td>xmFontListAppend/3</td>
<td>Append entries to a Font List</td>
</tr>
<tr>
<td>xmFontListCopy/2</td>
<td>Copy a Font List</td>
</tr>
<tr>
<td>xmFontListCreate/2</td>
<td>Create a new FontList from entry or list of entries</td>
</tr>
<tr>
<td>xmFontListFree/1</td>
<td>Free memory for a Font List</td>
</tr>
<tr>
<td>xmGetAtomName/2</td>
<td>Query atom name</td>
</tr>
<tr>
<td>xmGetCallbackComponent.../1</td>
<td>Retrieve components of a callback structure</td>
</tr>
<tr>
<td>xmGetDestination/1</td>
<td>Determine destination address for cut and paste functions</td>
</tr>
<tr>
<td>xmGetDragContext/3</td>
<td>Return widget name of DragContext active at Timestamp</td>
</tr>
<tr>
<td>xmGetFocusWidget/2</td>
<td>Determine focus widget within hierarchy of widget</td>
</tr>
<tr>
<td>xmGetMenuCursor/1</td>
<td>Query menu cursor currently being used</td>
</tr>
<tr>
<td>xmGetPixmap/4</td>
<td>Generate a pixmap</td>
</tr>
<tr>
<td>xmGetPixmapByDepth/5</td>
<td>Generate a pixmap</td>
</tr>
<tr>
<td>xmGetPostedFromWidget/2</td>
<td>Query menu environment</td>
</tr>
<tr>
<td>xmGetTabGroup/2</td>
<td>Return tab group widget of widget</td>
</tr>
<tr>
<td>xmGetTearOffControl/2</td>
<td>Return tear off widget of a menu</td>
</tr>
<tr>
<td>xmGetVisibility/2</td>
<td>Determine visibility status of a widget with respect to ancestors</td>
</tr>
<tr>
<td>xmGetXmDisplay/1</td>
<td>Return display widget of display</td>
</tr>
<tr>
<td>xmGetXmScreen/1</td>
<td>Return screen widget of screen</td>
</tr>
<tr>
<td>xmInternAtom/3</td>
<td>Register atom with the X server or check its existence</td>
</tr>
<tr>
<td>xmIsMotifWMRunning/1</td>
<td>Check status of the Motif Window Manager</td>
</tr>
<tr>
<td>xmIsTraversable/1</td>
<td>Identify whether widget can be traversed</td>
</tr>
<tr>
<td>xmListAddItem.../3</td>
<td>Insert entries in a list</td>
</tr>
<tr>
<td>xmListDelete.../1/2/3</td>
<td>Delete entries from a list</td>
</tr>
<tr>
<td>xmListDeselect.../1/2</td>
<td>Cancel markings on entries in a list</td>
</tr>
<tr>
<td>xmListGetKbdItemPos/2</td>
<td>Query position of a list item</td>
</tr>
<tr>
<td>xmListGetMatchPos/3</td>
<td>Query position of entries in a list</td>
</tr>
<tr>
<td>xmListGetSelectedPos/2</td>
<td>Query positions of marked entries in a list</td>
</tr>
</tbody>
</table>
xmListItemExists/2 – Check existence of an entry in a list .......... 299
xmListItemPos/3 – Query position of an entry in a list ............... 300
xmListPosSelected/2 – Check if list entry is selected ............... 301
xmListPosToBounds/6 – Query bounding box of a list item ........... 302
xmListReplace.../3 – Replace entries in a list ...................... 303
xmListSelect.../3 – Mark an entry in a list ......................... 305
xmListSetAddMode/2 – Set add mode for inserting entries into a list .... 306
xmListSetBottom.../2 – Specify last visible entry in a list ........... 307
xmListSetHorizPos/2 – Specify the position of the horizontal scroll bar .. 308
xmListSet.../2 – Specify the first visible entry in a list ............... 309
xmListUpdateSelectedList/1 – Update selected list .................. 311
xmListYToPos/3 – Query list item position at y-coordinate .......... 312
xmMainWindowSep.../2 – Specify the names of the separator widgets .... 313
xmMainWindowSetAreas/6 – Specify the children of the MainWindow widget 314
xmMenuPosition/1 – Position pop-up menu ............................ 315
xmMessageBoxGetChild/3 – Query a component of the MessageBox widget . 316
xmOption...Gadget/2 – Get information about a menu item .......... 317
xmProcessTraversal/2 – Specify the mode for processing keyboard events . 318
xmRemoveProtocolCallback/5 – Remove callback predicate from a protocol 319
xmRemoveProtocols/3 – Remove protocols ............................. 320
xmRemoveTabGroup/1 – Remove a widget from a tab group .......... 321
xmRemoveWMProtocolCallback/4 – Remove callback predicate from a WM protocol .................................................. 322
xmRemoveWMProtocols/2 – Remove WM protocols ...................... 323
xmRepTypeAddReverse/1 – Install reverse converter .................. 324
xmRepTypeGetNameList/3 – Generate list of values for a representation type 325
xmRepTypeGetRecord/2 – Generate list of data for a representation type .. 326
xmRepTypeGetRegistered/1 – Generate list of representation type names .. 327
xmRepTypeInstallTearOffModelConverter/0 – Install converter for Row-Column’s TearOffModel Resource .......................... 328
xmRepTypeRegister/3 – Register a representation type resource ........ 329
xmRepTypeValidValue/3 – Test validity of numerical value for representation
type resource ................................................................. 330
xmScale...Value/2 – Query or set position of the slider along a scale .... 331
xmScrollBar...Values/5 – Query or set the position of a scroll bar slider . 332
xmScrollVisible/4 – Make obscured widget or gadget visible ............. 333
xmScrolledWindowSetAreas/4 – Add or remove components of the Scrolled-
Window widget .................................................................. 334
xmSelectionBoxGetChild/3 – Query a component of the SelectionBox widget 335
xmSendClientMessage/5 – Test protocol facilities ............................. 336
xmSetCallbackComponents/1 – Set callback components for in/out members
of callback structures .......................................................... 337
xmSetFontUnit.../1/2 – Define the font size .................................... 338
xmSetMenuCursor/1 – Set menu pointer .......................................... 339
xmTargetsAreCompatible/2 – Test whether targets of drop site and source
are compatible ..................................................................... 340
xmTextClearSelection/1 – Cancel first marking in the Text widget ....... 341
xmText.../1 – Copy text into the Text widget’s clipboard .................... 342
xmTextDisableRedisplay/1 – Prevent visual update of Text widget ....... 343
xmTextEnableRedisplay/1 – Force visual update of Text widget .......... 344
xmTextField... – Predicates for processing a TextField widget ......... 345
xmTextFreeConstraints/1 – Cancel restrictions on user inputs into the Text
widget ................................................................................. 347
xmTextGetBaseline/2 – Query position of the first baseline in a Text widget 348
xmTextGetConstraints/5 – Query restrictions on user inputs into the Text
widget .................................................................................. 349
xmTextGetEditable/1 – Check that a text item in the Text widget may be
edited ................................................................................. 350
xmTextGetInsertionPosition/2 – Query the position of the insertion cursor
in the Text widget ............................................................... 351
xmTextGetLastPosition/2 – Query position of last character in the Text
widget .................................................................................. 352
xmTextGetMaxLength/2 – Query the maximum length of text in the Text widget 353
xmTextGetSelection/2 – Query marked text in the Text widget 354
xmTextGetSelectionPosition/3 – Query the position of marked text in the Text widget 355
xmTextGetString/2/3/4 – Query character string in the Text widget 356
xmTextGetSubstring/4 – Retrieve a portion of the text in a text widget 357
xmTextGetTopCharacter/2 – Query position of the first character in the Text widget 358
xmTextInsert/3 – Insert text into the Text widget 359
xmTextPaste/1 – Insert text from the Text widget clipboard 360
xmTextPosToXY/4 – Query the position of a character in the Text widget 361
xmTextRemove/1 – Delete text from the Text widget 362
xmTextReplace/4 – Replace text in the Text widget 363
xmTextScroll/2 – Move text in the Text widget 364
xmTextSetAddMode/2 – Set add mode for insertion of text into the Text widget 365
xmTextSetConstraints/5 – Specify restrictions on user inputs into a Text widget 366
xmTextSetEditable/2 – Specify whether a text item in the Text widget can be edited 367
xmTextSetHighlight/4 – Set highlighting for a text item in the Text widget 368
xmTextSetInsertionPosition/2 – Set the position of the insertion cursor in the Text widget 369
xmTextSetMaxLength/2 – Set the maximum length of character string in the Text widget 370
xmTextSetSelection/3 – Set text marking in the Text widget 371
xmTextSetSource/4 – Specify common sources for Text widgets 372
xmTextSetString/2 – Specify contents of a Text widget 373
xmTextSetTopCharacter/2 – Set the position of the first character in the Text widget 374
xmTextShowPosition/2 – Make text visible in the Text widget 375
xmTextPosToXY/4 – Query the position of a character in the Text widget 376
xmToggleButtonGadget...State/1/3 – Check and set status of the Toggle-Button gadget 377
xmTrackingEvent/4 – Select a widget 378
xmTrackingLocate/4 – Select a widget ............................................. 379
xmUpdateDisplay/1 – Initiate execution of expose events .................. 380
xmVersion/2 – Query the version of OSF/Motif .............................. 381
xmWidgetGetBaselines/2 – Retrieve Baseline information for a widget ... 382
xmWidgetGetDisplayRect/2 – Retrieve Display Rectangle information for a
widget .................................................................................. 383
xtAddAction.../1 – Add an action predicate ................................... 384
xtAddCallback.../1/4 – Add callback predicate to callback list ........... 386
xtAddEditresSupport/1 – Add support for editres client ..................... 388
xtAddGrab/3 – Restrict input to a particular widget .......................... 389
xtAddInput/5 – Set input predicate ................................................ 390
xtAddTimeOut/4 – Set timeout predicate ......................................... 392
xtAddWorkProc/3 – Set work predicate .......................................... 394
xtAugmentTranslations/2 – Augment Translations ............................. 395
xtBreakMainLoop/0 – Interrupt main loop ...................................... 396
xtCallAcceptFocus/1 – Determine readiness to accept keyboard focus .... 397
xtCallActionProc/2 – Call an action ............................................... 398
xtClass/2 – Output class and name of a widget ................................. 399
xtCreateApplicationShell/4 – Create toplevel window ....................... 400
xtCreateManagedWidget/5 – Create and manage child widget ............. 401
xtCreatePopupShell/5 – Create pop-up shell .................................... 403
xtCreateWidget/5 – Create widget .................................................. 405
xtDestroyWidget/1 – Destroy widget .............................................. 407
xtDisplay.../2 – Define width and height of a display ....................... 408
xtDrawArc.../2/7 – Draw circular or elliptical arcs ......................... 409
xtDrawBitmapExtents/4 – Obtain dimensions of a bitmap ................... 411
xtDrawClear/1 – Clear screen contents of Draw widget ...................... 412
xtDrawClearArea/6 – Clear a window area .................................... 413
xtDrawCopyArea/7 – Copy a window area ...................................... 414
xtDrawCopyBitmap/4 – Copy a bitmap .......................................... 415
xtDrawDeleteLine/5 – Delete a line .............................................. 416
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtDrawFillArc</td>
<td>Draw filled circular or elliptical arcs</td>
<td>417</td>
</tr>
<tr>
<td>xtDrawFillPolygon</td>
<td>Draw filled polygon</td>
<td>419</td>
</tr>
<tr>
<td>xtDrawFillRectangle</td>
<td>Draw filled rectangle</td>
<td>420</td>
</tr>
<tr>
<td>xtDrawFlush</td>
<td>Execute output operations immediately</td>
<td>421</td>
</tr>
<tr>
<td>xtDrawFontExtents</td>
<td>Query size of a font</td>
<td>422</td>
</tr>
<tr>
<td>xtDrawGetXY</td>
<td>Query last position of the pointer</td>
<td>423</td>
</tr>
<tr>
<td>xtDrawImageString</td>
<td>Output character string with foreground and background</td>
<td>424</td>
</tr>
<tr>
<td>xtDrawLine</td>
<td>Draw lines</td>
<td>425</td>
</tr>
<tr>
<td>xtDrawPoint</td>
<td>Draw points</td>
<td>426</td>
</tr>
<tr>
<td>xtDrawQueryDumpFile</td>
<td>Query size of a memory dump</td>
<td>427</td>
</tr>
<tr>
<td>xtDrawQueryPointer</td>
<td>Query position of the pointer</td>
<td>428</td>
</tr>
<tr>
<td>xtDrawReadWindowDump</td>
<td>Read a memory dump</td>
<td>429</td>
</tr>
<tr>
<td>xtDrawRecallGC</td>
<td>Set graphics context</td>
<td>430</td>
</tr>
<tr>
<td>xtDrawRectangle</td>
<td>Draw rectangles</td>
<td>431</td>
</tr>
<tr>
<td>xtDrawRepaint</td>
<td>Restore screen contents</td>
<td>432</td>
</tr>
<tr>
<td>xtDrawSegments</td>
<td>Draw line segments</td>
<td>433</td>
</tr>
<tr>
<td>xtDrawSetDashes</td>
<td>Set line pattern</td>
<td>434</td>
</tr>
<tr>
<td>xtDrawSetParam</td>
<td>Set parameters for the graphics context</td>
<td>435</td>
</tr>
<tr>
<td>xtDrawSetRubberGC</td>
<td>Set rubber-banding mode</td>
<td>438</td>
</tr>
<tr>
<td>xtDrawSetTSOrigin</td>
<td>Set origin for the pattern</td>
<td>439</td>
</tr>
<tr>
<td>xtDrawStoreGC</td>
<td>Save graphics context</td>
<td>440</td>
</tr>
<tr>
<td>xtDrawString</td>
<td>Output a character string</td>
<td>441</td>
</tr>
<tr>
<td>xtDrawStringExtents</td>
<td>Query the size of a character string</td>
<td>442</td>
</tr>
<tr>
<td>xtDrawUnsetRubberGC</td>
<td>Cancel rubber-banding mode</td>
<td>443</td>
</tr>
<tr>
<td>xtDrawWriteWindowDump</td>
<td>Create memory dump of the window contents</td>
<td>444</td>
</tr>
<tr>
<td>xtDrawXmString</td>
<td>Output a compound string</td>
<td>445</td>
</tr>
<tr>
<td>xtDrawXmStringImage</td>
<td>Output compound string with foreground and background</td>
<td>446</td>
</tr>
<tr>
<td>xtDrawXmStringUnderline</td>
<td>Output underlined compound string</td>
<td>447</td>
</tr>
<tr>
<td>xtFlush</td>
<td>Flush the output buffer of the X server</td>
<td>448</td>
</tr>
</tbody>
</table>
xtGetActionKeysym/2 – Get action cause .......................... 449
xtGetApplicationResources/3 – Retrieve values of resources .......... 450
xtGetApplicationNameAndClass/3 – Retrieve name and class of an application 452
xtGetDefault/3 – Retrieve Information from the resource database .... 453
xtGetDefaultColorInfo/2 – Retrieve color information .................. 454
xtGetEventComponent.../1 – Retrieve event component ............... 455
xtGetMultiClickTime/2 – Get multiclick Time ......................... 457
xtGetValue.../2 – Retrieve values of resources ....................... 458
xtGetWidgetMapState/2 – Determine whether widget is mapped ....... 460
xtGrabButton/9 – Reserve buttons and redirect button events .......... 461
xtGrabKey/3 – Reserve keyboard and redirect keyboard event .......... 463
xtGrabKey/6 – Reserve keyboard and redirect keyboard event ........ 464
xtGrabKeyboard/4 – Reserve keyboard ................................. 465
xtGrabPointer/7 – Reserve pointer and redirect pointer events ........ 466
xtHasCallbacks/2 – Check existence of callbacks .................... 468
xtInitialize/5/6 – Initialize Toolkit Intrinsics ......................... 469
xtInstall...Accelerators/2 – Specify accelerators to operate across windows 472
xtIsComposite/1 – Check that a widget belongs to the CompositeWidgetClass 473
xtIsManaged/1 – Check management of a widget ....................... 474
xtIsRealized/1 – Check that a widget is realized ...................... 475
xtIsSensitive/1 – Check that a widget can be activated ............... 476
xtIsSubclass/2 – Check class to which a widget belongs .............. 477
xtKeysymToKeycode/3 – Convert key symbol (keysym) to key code (keycode) and vice versa ........................................... 478
xtKeysymToKeycodeList/4 – Description ................................... 479
xtLoopWhileEvents/0 – Process outstanding events .................... 480
xtLowerWidget/1 – Place widget at bottom of window stack ............ 481
xtMainLoop/0 – Execute main event loop ................................. 482
xtManageChild.../1 – Manage child widgets ............................. 483
xtMapWidget/1 – Map a widget ............................................. 484
xtName/2 – Relate widget names to toolkit names ....................... 485
xtNameToWidget/3 – Convert widget name to window name
xtOverrideTranslations/2 – Override Translations
xtParent/2 – Find related parent and child widgets
xtPending/0 – Check for existence of events
xtPopdown/1 – Terminate output of a pop-up widget
xtPopup/2 – Output pop-up widget
xtPopupSpringLoaded/1 – Display pop-up widget
xtProcessEvent/1 – Process events
xtRaiseWidget/1 – Place widget at top of window stack
xtReInitialize/0 – Release widgets
xtRealizeWidget/1 – Realize widget
xtRemoveAllCallbacks/2 – Remove all the callback predicates in a callback list
xtRemoveCallback.../1/4 – Remove callback predicates
xtRemoveGrab/1 – Remove input restrictions
xtRemoveInput/1 – Cancel input job
xtRemoveTimeOut/1 – Cancel timeout job
xtRemoveWorkProc/1 – Cancel work job
xtRestackWidget/3 – Change window stacking order
xtSetKeyboardFocus/2 – Redirect input events
xtSetLanguageProc/0 – Add internationalisation support
xtSetMappedWhenManaged/2 – Output widget
xtSetMultiClickTime/2 – Set multiclick time
xtSetSensitive/2 – Set widget so it can be activated
xtSetUserSpecifiedLocation/1 – Description
xtSetValue.../2 – Set and modify resources
xtSetWidgetCursor/2 – Set cursor for a widget
xtSync/1 – Flush output buffer
xtTopLevelShellWidget/1 – Output widget name of the toplevel widget
xtTranslateCoords/5 – Convert coordinates
xtUngrabButton/3 – Cancel button reservation
xtUngrabKey/3 – Cancel keyboard reservation by a key ............... 518
xtUngrabKeyboard/1 – Cancel keyboard reservation .................. 519
xtUngrabPointer/1 – Cancel pointer reservation ....................... 520
xtUnmanageChild.../1 – Switch off management of a widget ........ 521
xtUnmapWidget/1 – Cancel mapping of a widget ....................... 522
xtUnrealizeWidget/1 – Delete window for a widget ................... 523
xtUnregisterWidget/1 – Unregister a widget ......................... 524
xtVersion/2 – Query the version of X Window System ............... 525
xtWidget/1 – Query widget names .................................... 526
xtWindowToWidget/2 – Query window id or widget name ............ 527

A Programming examples ........................................... 529
A.1 Example of an editor using various OSF/Motif functions .......... 529
A.2 Example showing communication with the Motif Window Manager .... 541
A.3 Pop-up menu example ........................................... 543

B Callback structure components ................................ 547

C Exceptions ......................................................... 557

D Glossary .......................................................... 567

Bibliography ........................................................... 579

Index ................................................................. 581
Chapter 1

Preface

The IF/Prolog system from Siemens AG Austria is an implementation of the ISO Prolog standard (ISO = International Standardization Organization). This standard was prepared by ISO Working Group 17, comprising representatives from various national standardization bodies.

IF/Prolog also contains interfaces and predicates which extend the language and ensure compatibility with earlier versions of the product.

The Standard dictates us to supply a strictly conforming mode, where the Prolog system only accepts and supplies conforming language predicates. To invoke this mode, see the section on invoking IF/Prolog in the User’s Guide.

The IF/Prolog OSF/Motif Interface is a component of the IF/Prolog system. It is a programming interface with which applications and interfaces can be implemented as under OSF/Motif.

Target group

This manual is intended for anyone working with IF/Prolog who wishes to create programs with graphical user interfaces (GUIs).

The manual does not include a comprehensive description of OSF/Motif or the X Window System. Details of these will be found in the relevant original documentation.

Manuals

The documentation for IF/Prolog comprises of the following manuals:

- IF/Prolog Reference Manual
- IF/Prolog User’s Guide
Preface

- IF/Prolog Windows Interfaces
- IF/Prolog OSF/Motif Interface
- IF/Prolog Constraints Package

The IF/Prolog Reference Manual contains a description of the semantics, built-in predicates, C interface functions, debugger commands and environment control of IF/Prolog. It also contains an overview of the syntax of the Prolog language.

Predicates associated with an interface or package are described in the respective manuals. The IF/Prolog User’s Guide describes how to work with Prolog on a computer running under the UNIX, DOS, Windows and other operating system.

As the set of manuals for IF/Prolog are not tutorials, you should be familiar with the Prolog language. You should also be familiar with the basics of the operating system and know how to use one of the editors installed on your computer.

The IF/Prolog OSF/Motif Interface Manual contains details of all the elements of OSF/Motif which can be defined in IF/Prolog and specifications of all the predefined ("built-in") predicates available for that purpose.

Notational conventions

The following notational conventions are used throughout this manual:

- Syntax definitions are enclosed within a frame.
- Prolog language elements, operating system commands and outputs from the system are printed in teletype font.
- Italics are used to represent variable parts in inputs and outputs where you should substitute them with your own values.
- Square brackets denote optional entries in the syntax notation; the brackets are not part of the Prolog text.
- Square brackets in bold type are elements of the Prolog list notation and are part of the Prolog text.
- Braces denote alternatives in the syntax notation; the brackets are not part of the Prolog text.
- A bar denotes alternatives in the syntax notation.
- Parentheses are required parts of the Prolog predicate notation and is part of the Prolog text.
- Ellipsis indicate that the preceding syntax element may be repeated.
- Predicates are specified in the form Name/Arity.
- Several predicates with the same name and different arities are denoted in this form.
- A number in square brackets indicates a reference to another manual or a textbook. The number identifies the publication in the Bibliography at the end of the manual.
The following pictograms are also used:

- **i** for important advice and related information.
- **!** for warnings.
Preface
Chapter 2

Using IF/Prolog for OSF/Motif programming

This chapter provides some basic details about the programming of the IF/Prolog interface for OSF/Motif. In this, particular emphasis is given to the differences between the IF/Prolog interface and the C interface.

This description covers:

- The basic concepts of X and OSF/Motif programming
- The differences between using IF/Prolog and using C to program OSF/Motif
- The structure of an OSF/Motif application
- The User Interface Language UIL

This chapter gives merely an introduction to the programming of OSF/Motif. For more detailed information, the reader should refer to the original literature on Xt Intrinsics and OSF/Motif ([14] through [21]).
Introduction to the X Window System and OSF/Motif

The X Window System, or more briefly X, is an elementary system for the operation of graphic data display terminals, and for the development of applications which use these devices. It allows several applications, each using a number of windows, to be operated simultaneously from the same screen.

Clients and Server

Under the X Window System, applications are known as clients. X clients can be programmed independently of the hardware, because the access to the hardware is implemented by the X server. The client passes specific tasks to the server (e.g. to output a window on the screen, or to draw a graphic element) and the server executes these in accordance with its knowledge of the hardware for which it is responsible. The hardware for which any server is responsible will comprise at least a screen, a keyboard, and a graphic input device - generally a mouse. The screen, keyboard and mouse are referred to collectively as the display. Each server handles one display, and may serve several clients.

The window manager is a special X client. Its purpose is to manage the individual clients’ windows on the screen, i.e. to determine their size and position, and to alter these when necessary (e.g. reduce a window to an icon when it is not currently required). The window manager most frequently used with OSF/Motif is the Motif Window Manager (mwm).

OSF/Motif and the X Window System

The X Window System and OSF/Motif provide several libraries containing C functions, for use in creating X applications:

- Xlib containing basic functions of the X Window System (in particular, graphics and communication functions),
- X Toolkit Intrinsics (Xt Intrinsics), with routines for the implementation and general manipulation of complex objects. Compared to Xlib, this represents a higher level of abstraction. However, Xt Intrinsics does not provide any concrete objects, but merely the means of working with them.
- The OSF/Motif widget set builds on the Xt Intrinsics and defines a set of object types which are actually available (e.g. menus and buttons); it also provides functions for use in designing graphical interfaces.

None of the three components supplies ready-made interfaces, but merely tools with which to develop them. In addition to its comprehensive set of widgets, for use in displaying windows and control elements, OSF/Motif also provides the programmer with a range of applications (clients). The most important of these are the Motif Window Manager (mwm) and the User...
Interface Language (UIL) - an interface description language which enables the appearance of a user interface to be stored as a specification in a file.

OSF/Motif also includes a collection of guidelines for the development of an OSF/Motif user interface, the *OSF/Motif Style Guide* [21].

**Basic programming concepts for OSF/Motif**

There now follows a brief explanation of the most important of the basic concepts of the X Window System and OSF/Motif.

**Widgets**

Widgets are complex objects, each consisting of a window and associated functions. Apart from information about its appearance, a widget contains all the functions which it can execute.

Examples of widgets are: buttons, scroll bars, and windows for text editing.

Widgets are divided according to their functionality into:

- **Display widget**
  These are control elements which have functions for user inputs or outputs from the program, e.g. buttons. They cannot have any child widgets.

- **Container widgets**
  These are higher-level widgets, e.g. menus, which manage the position and size of their child widgets on the screen, e.g. menus.

- **Shell widget**
  These are special Container widgets, which establish the communications with other components of the X window interface, in particular with the Window Manager. A Shell widget can only manage one child widget. They remain virtually invisible, as they have widget children which completely cover them.

An OSF/Motif application is implemented by means of a hierarchy of widgets (see figure 2.1, page 11), the root of which is a Shell widget. This is called the TopLevel widget. Beneath this lies a Container widget, which is in turn responsible for the management of further Container or Display widgets. The Display widgets implement the actual functions of the application.
**Resources**

Widgets can have a large number of properties; these affect their appearance and behavior. The properties are called resources. Each resource has a name and a value. The values have default settings for the widgets. These resource values can be modified in various places:

- in the program itself
- when the application is called, by specifying options
- in configuration files

The external predefined values for the resources are held in a global resource database, which is managed by a Resource Manager.

**Widget classes**

A widget class describes the properties (resources) of a number of widgets of the same type. The widget class embodies the basic functionality of the widgets. The widgets created from a widget class are referred to as widget instances. For these widget instances, the applicable resources may either retain the predefined values for the class or be given values which differ from these.

To allow different classes to access common resources, a class hierarchy is formed. The parameters for each widget are the resources defined for its own class, together with the resources for the superclass(es) of its class. The widget hierarchy is shown in figure 2.1. (page 11)

The Xt Intrinsics functions provide a number of classes which define fundamental resources. These classes are opaque widget classes; that is to say, they themselves are never instantiated, but serve only to pass down resources to those widget classes which can be instantiated. They are used only as a starting point for the implementation of "real" widget classes.

The Intrinsics functions can be used to set up the various Shell widgets which it is possible to instantiate.

The following are the opaque widget classes:

- `widgetClass` see the Core widget class
- `compositeWidgetClass` see the Composite widget class
- `constraintWidgetClass` see the Constraint widget class
- `shellWidgetClass` see the Shell widget class

The following are the shell classes:
OverrideShellWidgetClass  see the OverrideShell widget class
vendorShell  see the VendorShell widget class
wmShellWidgetClass  see the WMShell widget class
transientShellWidgetClass  see the TransientShell widget class
topLevelShellWidgetClass  see the TopLevelShell widget class
applicationShellWidgetClass  see the ApplicationShell widget class

The widget classes are described in detail in the chapter *Widgets and gadgets* (page 35).

**Gadgets**

Gadgets have the same functions as the widgets of the same names. However, they do not have windows of their own; instead, they are added to the windows of their parent widgets. The parent widget also manages the events for the gadget. A gadget cannot have color specifications of its own, nor any translation or accelerator definitions (see below).

**Events and callbacks**

A widget can be linked with certain functions, which are called when a particular event occurs. Events are generated by the X server. They are mostly caused by user actions (e.g. moving the mouse or pressing a key) or a change in state of the windows of an application (e.g. if a window which was previously hidden is exposed once again). However, they may also be generated in a program.

The event loop for the application processes the events in sequence. The events are passed on to the widgets. Widgets can either handle the events themselves or pass them as so-called "callbacks" to the application for processing.

Callback lists (which are special widget resources) contain the functions of the application; these will be called when an event or a condition occurs which is defined as the trigger for the callback concerned. The correspondence between events and callbacks is partly predefined by the widgets and partly configurable by means of translations and actions.

**Translations and actions**

The functions of the widgets are called as a reaction to events. The event which triggers each function is defined for each widget in two tables, the translation table and the action table.

The translation table determines which sequence of events entered by the user (e.g. a particular sequence of keys) triggers which action. The action table determines which action calls which function of the application or widget.

The following is an example of the sequence of activities from a user input through to the execution of a desired action.
Event: User presses mouse button
Translation table: \(< Btn1Down >\): arm()
Action table:
  arm: arm (...) 
  Process the callback list for armCallback

The translation table for a widget can be modified. That is, a programmer or user can use the translations resource of a widget to specify, for example, that pressing mouse button rather than mouse button is the event which is to call the action arm().

The action table for a widget cannot be modified. However, additional actions may be defined, and called via the translation table.

A callback is specific to a particular widget, and is triggered by the appropriate change in state of that widget. Actions, by contrast, are global, i.e. independent of any specific widget; they can appear as actions in the translation table of a number of widgets.

**Accelerators**

An accelerator (shortcut selector) can be used to link an action that is executed within one widget to events in another widget. For example, input by the user of a particular key sequence within one widget could initiate the activation of another widget, say, a menu button. This speeds up (accelerates) user inputs.

The binding of events to actions is implemented in the accelerator table, in a analogous manner to the translation table.
OSF/Motif 1.1 Widget-Set

Figure 2.1: The widget hierarchy in OSF/Motif
OSF/Motif programming using IF/Prolog

The IF/Prolog predicates make available the functionality of Xt Intrinsics, plus the additional widgets and functions provided by OSF/Motif.

Programming is further simplified by an additional widget, the Draw widget, which provides almost all of the drawing functions of Xlib. Furthermore, other important Xlib functions such as sequencing window stacks, or querying the dimensions of a screen, are implemented by special IF/Prolog predicates.

The external functions of the widgets are also implemented as IF/Prolog predicates, with similar arguments. The interface checks whether the widget which has been passed to the external functions belongs to a subclass of the class which provides the function.

A crucial advantage of the interface is the ability to use IF/Prolog data types when functions of OSF/Motif or Xt Intrinsics are used. Data types used in IF/Prolog are different than in C, and hence an automatic conversion is carried out in most cases.

Callback predicates in IF/Prolog correspond to the callback functions in C. They have three arguments which exactly correspond to those in C.

Differences between the IF/Prolog OSF/Motif Interface and the OSF/Motif C interface

- The names which are used are the same as in OSF/Motif, except that the initial uppercase letters are converted to lowercase, because IF/Prolog assumes that names which start with an uppercase letter are variables.

- In OSF/Motif, macros are often used as synonyms for numeric constants. In IF/Prolog, atoms made up of an equivalent character string are used instead. Enumerated types are treated in exactly the same way.

- In IF/Prolog, widgets and classes are not represented by pointers, but by names (atoms). All widgets that are created are given special IF/Prolog names. When referring to widgets from within IF/Prolog, these IF/Prolog names must be used. The special name `none` is reserved for the C value `NULL` (no widget).

- Callbacks cannot be specified as resources when a widget is created in IF/Prolog (using `xmCreate...`), but must be added after the widget has been created, using `xtAddCallback(s)`.

- In the case of the callback component `value`, it is necessary in some callback structures to distinguish whether the value expected is an XmString or an integer. In IF/Prolog, the atom specifying the callback component must be strictly bound to a particular type. A distinction must therefore be made between `value` for integers and `valueString` for XmStrings.
• The following resources cannot be used in IF/Prolog, because processing or representation of the argument values is not possible:

```
argv           insertPosition
argc           qualifySearchDataProc
colormap       screen
createPopupChildProc source
dirSearchProc  visual
fileSearchProc
```

• Direct event handling is not offered in IF/Prolog. Translation and action tables and callbacks must be used instead. These represent a higher-level interface to events. It is not possible in IF/Prolog to define event handlers which allow events to be handled at an elementary level.

• The Draw widget gives the user a simple way of accessing the drawing operations of Xlib. This widget automatically saves all drawing operations, and then re-executes them if an expose event occurs, i.e. if part of the widget is exposed.

• In cases where the data type `array` is used in Xt Intrinsics or OSF/Motif, IF/Prolog uses the data type `list`.

• In IF/Prolog it is very much easier than in C to use variable data items as arguments. Conversion of Xt Intrinsics argument lists to IF/Prolog lists therefore greatly simplifies programming.

• Arguments are checked automatically. If invalid arguments are used, IF/Prolog exceptions will be generated.

• Resource names in C are defined as macros with string values. In IF/Prolog, these can be used directly as atoms.

• Depending on their type, resource values in IF/Prolog are represented by different terms. Some resource values cannot be converted back to symbolic representation when they are queried using `xtGetValue` (this is also true in C). For such values, IF/Prolog atoms are generated which allow the values to be passed on to other widgets (e.g. font lists and color, among others).

• In general, the parameters used in IF/Prolog are the same as in C. Counters which only record the lengths of variable length lists are omitted. The required length is calculated automatically by the system.

• Function results are normally returned via the first argument of the appropriate predicate.
Compound strings and font lists

Compound strings are character strings which are made up of various components. This allows language-independent character strings to be encoded. The components of a compound string are:

- CharSet: Symbolic character set name
- Direction: Direction in which characters are to be output
- Text: Text
- Separator: Line separator

These components can be used to define multi-line character strings using various character sets and written in different directions. Compound strings have their own data type, XmString (see page 16).

Font lists are tables which assign symbolic character set names to their associated character sets. The names are not converted into the actual fonts until the time of output. The data type of a font list is XmFontList (see page 17).

The IF/Prolog representation of compound strings and font lists is used to set widget resources using xtSetValue, and to retrieve widget resources using xtGetValue. In addition they are used as arguments of predicates which correspond to the external functions of a particular widget class.

The string format which is used in resource files for font lists and strings is also permitted for passing data from IF/Prolog to a predicate. In this case, atoms are used.

Callback structures

In OSF/Motif, all callbacks are called with a data structure containing data which will be of interest to the application at a later time. This data is variable, depending on why the callback arose. The components will often be a constant with the name reason, and an event structure. The constant reason together with the widget class will determine which data is passed. The event structure contains information about the event which led to the callback.

In the case of the IF/Prolog OSF/Motif Interface, only the reason for the callback is passed, as an atom, to a callback predicate; this will be the last argument. The reason for the callback is converted to a character string by a converter, which translates all predefined callback reasons.

The other components of the callback structure must be interrogated using a special predicate, and for this reason they are only converted to their IF/Prolog representation on request. In most cases, an application will not need to query the components of the callback (xmGetCallbackComponent(s)/1) because it can obtain the same information directly from the widget (xtGetValue(s)/2).
**Cut and paste functions**

The functions which transfer data between an X Window clipboard and an application are also available in the form of IF/Prolog predicates.

Here, there are currently still some restrictions on the data formats. The formats specified in [20] *ICCCM Inter-Client Communication Conventions* usually cannot be used, as the corresponding converters from IF/Prolog to C have not been implemented.

IF/Prolog has two predefined formats of its own, with corresponding converters:

- `prolog_atom` for atoms
- `prolog_atom_list` for lists of atoms

All other format names (including user defined ones) allow only atoms to be specified in IF/Prolog, and these are converted to strings in C.

Display and window parameters for functions are taken over from the widget parameters. The conversion routine computes the length of the data from the data items.
OSF/Motif and IF/Prolog

Restrictions

- Applications which use more than one display are not possible.
- The maximum number of global actions is 10.

Error handling

The Xt Intrinsics functions react categorically to any error situation. For example, IF/Prolog will be terminated and an error message output if the X server is terminated while the OSF/Motif application is running.

The possible error situations are not fully described in X, and therefore no general means of intercepting them is available in IF/Prolog.

One error situation which can typically arise during interactive dialogs is checked. If it is impossible to open the appropriate display during `xtInitialize/5`, the exception `cannot_open_display` is created.

Error messages issued by X Toolkit cannot be suppressed.

Terminating IF/Prolog

If it is required to terminate IF/Prolog in the middle of an OSF/Motif run, the QUIT signal must be used. This interrupts the program immediately and terminates IF/Prolog.

OSF/Motif data types in IF/Prolog

For resources and the arguments of functions, the X Window System and OSF/Motif define a number of special data types. These are mapped to IF/Prolog data types on the basis of the corresponding resource tables or predicate specifications. For some complex data types, the content must also be specified. This section explains the syntax and semantics of such complex data objects in IF/Prolog.

XmString

XmString is the data type for compound strings. These are represented in IF/Prolog as lists made up of the following elements:

text = Text or text(Text )
charset = CharSet or charset(CharSet )
direction = Direction or direction(Direction )
separator
Text
For XmString components either assignment or predicate notation can be used. E.g. an XmString component can be either entered as \texttt{text=value} or \texttt{text(value)}. Both syntactic variants can be mixed in the same XmString specification.

\begin{verbatim}
[ Text1, Text2, ... ]
\end{verbatim}

Text Atom, a visible character string  
CharSet Atom, the symbolic name of a cset or the constant string \texttt{default_charset} 
Direction Atom, either \texttt{string\_direction\_l\_to\_r} or \texttt{string\_direction\_r\_to\_l}

\textbf{Example}

\begin{verbatim}
[  
   charset = chset1,  
   direction = string\_direction\_l\_to\_r,  
   text = text1,  
   separator,  
   text = text2  
]
\end{verbatim}

When specifying a simple atom this corresponds to a compound string with a text component, the default character set and the default direction.

\textbf{XmFontList}

In IF/Prolog, font lists are represented as lists with the following elements:

\begin{verbatim}
[ FontName = CharSet, ... ]
\end{verbatim}

FontName Atom, the name of one of the server’s available fonts  
CharSet Atom, an arbitrary symbolic name which is used in a compound string

It is also possible to specify an atom as the font for the default character set.
Example

\[
[ \ '12x24' = \text{chset0}, \\
\ '\ast\text{bold}\ast\text{-o}\ast\text{-14}\ast\text{m}\ast' = \text{chset1} \ ] \ \text{or} \\
\ 'gs \ 24'
\]

Note
When they are interrogated, resource values of this type cannot be converted back to a symbolic representation. For these values, a unique atom is returned and can be used in subsequent processing.

XmStringTable

Tables of XmStrings are represented in IF/Prolog as lists of XmStrings (see above).

XtTranslation

Translation tables are represented in IF/Prolog as atoms; the text of the atom must satisfy the syntax rules of Xt Intrinsics (cf. [16]).

Example

\['\#\text{override}\n\ <\text{Key}> \text{ Up: UpOrDown(0) } \n\ <\text{Key}> \text{ Down: UpOrDown(0)}',
\]

or

\['\#\text{override}\n <\text{Key}> \text{ Up: UpOrDown(0) } \n <\text{Key}> \text{ Down: UpOrDown(0)}',
\]

Note
When they are interrogated, resource values of this type cannot be converted back to a symbolic representation. For these values, a unique atom is returned and can be used in subsequent processing.

XtAccelerator

XtAccelerators are represented in IF/Prolog as atoms; the text of the atom must satisfy the syntax rules of Xt Intrinsics (cf. [16]).

Note
When they are interrogated, resource values of this type cannot be converted back to a symbolic representation. For these values, a unique atom is returned and can be used in subsequent processing.
Pixel

This data type identifies a color value. In IF/Prolog, the color name can be specified as an atom.

*Note*

When they are interrogated, resource values of this type cannot be converted back to a symbolic representation. For these values, a unique atom is returned and can be used in subsequent processing.

Window

Some functions use or supply the identifier of a window, as used at the Xlib level (X Window System). Every widget that has been realized is linked to such a window. This linkage can be determined using the predicate `xtWindowToWidget/2`.

Pixmap

In IF/Prolog, each pixmap is identified by the name of the file which contains its definition. A particular method is used in searching for this file. For this search, pixmap file names can be specified either absolutely (`'/<path>'` or `'./<path>'`) or relative to a global resource `bitmapFilePath`. If the value of this resource is not set either in the resource file or with the help of command arguments (`-xrm`), then a system-specific default value will apply (generally `/usr/include/X11/bitmaps`).

*Note*

When they are interrogated, resource values of this type cannot be converted back to a symbolic representation. For these values, a unique atom is returned and can be used in subsequent processing.
Programming procedure

Structure of an OSF/Motif application

The following section shows the structure of a typical OSF/Motif application. In this, the following steps are described:

- Initializing the X environment
- Creating and mapping widgets
- Processing user events
- Modifying resources

At the end of the section, a simple example is used to illustrate the structure of an application.

Initializing the X environment

To make the objects from Xt Intrinsics available, the X environment must be initialized. This is done using the predicate `xtInitialize/5`.

In IF/Prolog, it is not possible at the time when a program is started to determine whether and where an X server is available. From within X, it is possible to specify a display as an argument of the predicate `xtInitialize/5`, and this only becomes relevant at run time. Hence:

The specification of the display to be used will be evaluated as late as possible, i.e. not until `xtInitialize/5`. Only at that point is it possible to determine whether the link to the server can be established, or whether the Toolkit is reporting an error.

The option `-display <display:<x>.<y>>` will be evaluated by X if it is passed as follows:

```
user_parameters( Pars ), xtInitialize( ..., [Title|Pars]).
```

Here, however, a user parameter is involved, not a system parameter (cf. chapter entitled The IF/Prolog system, section on Calling IF/Prolog in the IF/Prolog User Guide [2]). The evaluation and any error messages issued are purely a matter for X Toolkit.

Creating and mapping widgets

After the initialization phase, a tree-like hierarchy of widgets is built up. This takes place in two steps:

- the widget tree is created.
- the widget tree is realized.
Creating the widget tree

The creation of the widget tree determines the hierarchy of the widgets. The first requirement is to create a Shell widget as the toplevel widget. Then beneath this, further widgets are created. Widgets are created using the predicates `xtCreateWidget/5`, `xtCreateManagedWidget/5`, or the corresponding predicates `xmCreate.../4`.

As Shell widgets may only have one child widget, the latter should be a Container widget, to permit branching in the hierarchy tree. Beneath the Container widget which is a child of the toplevel Shell, further Container or Display widgets can be added to the widget tree.

When a child widget is created, it is not automatically taken into account in the layout at this stage. The only effect is to provide the necessary data structures. The child widget must be passed to the parent widget for management, so that it can then also be mapped. This step is effected by using the predicate `xtManageChild/1`. The predicate `xtCreateManagedWidget/5` creates a child widget and at the same time passes it to the parent widget for management.

Realizing the widget tree

After the widget structure has been built, it can be realized using the predicate `xtRealizeWidget/1`; that is, the size and position of all the widgets can be defined, the windows created and the tree mapped.

Possible states of a widget

- created: A data structure has been created and added to a widget tree.
- managed: It has been passed to the parent widget for layout management.
- realized: The associated window has been created.
- mapped: The window is visible.

Processing user events

When the widget tree has been created, the predicate `xtMainLoop/0` can be called. This starts the main event loop, and interaction with the user can begin. This loop is an endless one, in which all events are received, and processed or passed on to the appropriate widget for processing. The actual functionality of the application is then implemented by callback predicates. `xtBreakMainLoop/0` can be used to interrupt the event loop from within a callback predicate.

If `xtLoopWhileEvents/0` is used instead of `xtMainLoop/0`, the event loop will only be processed as long as events are present.

The following example handles the events `EnterNotify` and `LeaveNotify`:
Example

```prolog
:- import( motif ).

program :-
  program_parameters(Pars),
  xtInitialize(toplevel, toplevel, herbie, [ ], Pars).
  xtAddActions( [enter, leave ]),
  xmCreateLabel( Label, toplevel, label, [
    labelString = mousesensitive,
    width = 100,
    height = 300]),
  xtSetValues( Label, [ translations =
    '#override
    <Enter>: enter() 
    <Leave>: leave() 
    ']),
  xtManageChild( Label),
  xtRealizeWidget( toplevel),
  xtMainLoop,
  write(quitting),
  nl.

  enter( _, _ ) :-
    write( enter ),
    nl.

  leave( _, _ ) :-
    write( leave ),
    nl,
    xtBreakMainLoop.
```

If the mouse cursor is moved across the widget, it will be seen that the events are processed as expected.
Setting and modifying resources

There are two ways in which the values for resources can be set within an application:

- When the widget is created, using the argument ArgList
- Using the predicates xtSetValue/2 or xtSetValues/2

The form in which the resource values are specified is:

- \[ \text{ResourceName} = \text{Value}, \ldots \] or \[ \text{ResourceName}(	ext{Value}), \ldots \] for ArgList and xtSetValues/2.
- ResourceName = Value or ResourceName(Value) for xtSetValue/2.

The predicates xtGetValue/2 and xtGetValues/2 are used to query the resource values for a specific realized widget.

Resources can be specified in either assignment or predicate notation. Both syntactic variants can be mixed in the same resource list.
Sample program

The following example shows a simply-structured *IF/Prolog OSF/Motif Interface* application.

```
/*********************************************************************************
* ( c ) Copyright 1994 Siemens Nixdorf Informationssysteme AG *
*********************************************************************************
* This dialog program allows a text item to be input into a Text widget. Clicking the Confirm button causes the string to be read from the widget and output. *
* *
* - Initialization of Intrinsics, setting up of a toplevel Shell and a Container widget. *
* - Setting up of the Text widget and the Confirm and Cancel buttons. *
* - Definition of the callback predicates and the Confirm and Cancel actions. *
*********************************************************************************/

:- import( motif ).

program :- user_parameters( Parameter ),
    % User parameters are stored in the Parameter list
    xTInitialize( Toplevel, Toplevel, dialog, [],
        [dialog | Parameter]),

    % Tasks of xTInitialize:
    % - Set up links to the server
    % - Initialize the Xt Toolkit
    % - Create a toplevel Shell
    % - Allocate resources

    % Explanation of the individual arguments:
    % The Shell name is assigned by IF/Prolog, because it is specified as a variable, 'Toplevel'. The Shell name is here also used as the Toolkit name. The name chosen in this case for the application class name is 'dialog'. The options list is empty.
    % The first component in the argument list is 'dialog', and this is used as the title. The other components of the argument
Programming procedure

% list are the components of the user parameter list 'Parameter'.

xtSetValues( Toplevel, 
    [allowShellResize = true, 
    input = true 
] ),

% This sets the resources allowShellResize and input for 
% the toplevel Shell. The resource value allowShellResize = true 
% allows the toplevel Shell to adjust when there are changes 
% in the size of its child, or that child’s descendants.

% The resource value input = true would also be set by default. 
% It has the effect that mwm gives the keyboard focus to the 
% application window.

xtAddAction( action ),
% Here, action is declared as a global action.

xtCreateManagedWidget( dialog, dialog, xmFormWidgetClass, 
    Toplevel, [ ] ),

% Here, a Form widget is created as the child of the toplevel 
% Shell widget. The Form widget is a Container widget. It 
% manages the layout of its children with the help of 
% attachments. These are resources of the Form widget, but are 
% set for its children.

xtCreateManagedWidget( Label, Label, xmLabelWidgetClass, dialog, 
    [labelString = 'Hello, please type input:' 
] ),

xtCreateManagedWidget( input, input, xmTextWidgetClass, dialog, 
    [topAttachment = attach_widget, 
    topWidget = Label 
] ),

xtCreateManagedWidget( Confirm, Confirm, xmPushButtonWidgetClass, 
    dialog, 
    [labelString = 'Confirm', 
    topAttachment = attach_widget, 
    topWidget = input 
] ),

xtCreateManagedWidget( Cancel, Cancel, xmPushButtonWidgetClass, 
    dialog, 
    [labelString = 'Cancel', 
    topAttachment = attach_widget, 
] ),
Programming procedure

leftAttachment = attach_widget,
topWidget = input,
leftWidget = Confirm ] ),

% The widget tree has now been completely set up, producing the
% following tree structure for the dialog application:

%  
% Toplevel (ToplevelShell)
%    |    
%    |    
%    --------------
%    ----------------| dialog |---------------
% | |(Form) | |
% | -------------- |
% | | | |
% | | | |
% Label input Confirm Cancel
% (Label) (Text) (PushButton) (PushButton)

% In addition, callbacks have been added in to the
% activateCallback lists of the two PushButtons:

xtAddCallback( Confirm, activateCallback, confirm, input ),
xtAddCallback( Cancel, activateCallback, cancel, _ ),

% The fourth argument of xtAddCallback/4 is a parameter which
% is passed to the callback predicate as the second argument.
% Hence, the name of the Text widget input is passed to the
% callback predicate 'confirm'; on the other hand, the callback
% predicate 'cancel' requires no information, and so the
% anonymous variable is used for the parameter.

xtSetValues( input,
    [translations = '#override
         <Key>Return: action( confirm )
         Ctrl <Key>C: action( cancel )'
    ] ),

% This defines translations for the Text widget input:
% specifying override will result in any existing translations
% being overwritten.

% When an action is called, it is also possible to pass a string
% as a parameter: action(string).
% The action itself should be defined as a predicate with an
% arity of two,

% action(Widget, params(second_argument)) :- ..... 

% A string passed with the call corresponds to the second
% argument in the action.

% There are two possible applications for this:

% 1) second_argument is a variable, in which case it will
   % be unified with string.
   % Example: action( Widget, params( Variable ) ) :- ...

% 2) second_argument is an atom.
% PROLOG then searches for the definition of the action for
% which the string matches the atom. This action is then
% executed. This allows different functionalities to be
% linked to a single action name.
% This is very useful, since it is only possible to define
% 10 global actions per application.

% Example (see also further below in the program text):

% action( Widget, params( confirm ) ) :-
%   xtGetValue( input, value = [ String ] ), !,
%   write( 'String "' ), write( String ),
%   write( " entered\n' ).
% action( Widget, params( cancel ) ) :-
%   halt.
% The second strategy has been used in this sample program.

% It will be noted that in this program the same functionality
% has been implemented by means of two different mechanisms:
% If the Confirm PushButton is pressed, then the callback
% confirm is called, and this contains exactly the same
% instructions as the action action(Widget, params(confirm)),
% which is triggered by pressing the return key with the Text
% widget activated. Similar behavior results from using the
% Cancel PushButton with its associated callback cancel and
% from the action action(Widget, params(cancel)), if the
% input <Ctrl> C is made into the Text widget.

xtRealizeWidget( Toplevel ),
% Realizing a widget creates the associated X window. A widget
Programming procedure

% which has been realized is not mapped until all its
% ancestors in the widget tree have also been realized. If a
% widget is realized, then all its children will also be
% realized.
% Consequently, one call of xtRealizeWidget for each toplevel
% widget in the application is sufficient.

xmProcessTraversal( input, traverse_current ),
xtMainLoop.

% Starts event processing by entering the main event loop,
% xtMainLoop. xtMainLoop calls up XtNextEvent and
% XtDispatchEvent alternately.
% XtNextEvent fetches the events from the X event queue, i.e.
% XtNextEvent reads an event and removes it from the queue.
% XtDispatchEvent presents the event to those widgets to which
% it is of interest. An event is of interest to a widget if the
% latter has a corresponding entry in its translation table, or
% it has a matching callback.

% 'confirm' obtains the string which was input, and outputs it.

confirm( _, Input, _ ) :-
    xtGetValue( Input, value = [ String ] ), !,
    write( 'String "' ), write( String ), write( '" entered
' ).
confirm( _, _, _ ).

action( _, params( confirm ) ) :-
    xtGetValue( input, value = [ String ] ), !,
    write( 'String "' ), write( String ), write( '" entered
' ).
action( _, params( confirm ) ).

action( _, params( cancel ) ) :-
    halt.

% cancel terminates the program.

cancel( _, _, _ ) :-
    halt.

Further sample programs dealing with some of the problems of programming OSF/Motif
using IF/Prolog will be found in the appendix. Source code for these and other examples
will be found in the directory $PROROOT/DEMOS/motif.
This section presents the User Interface Language (UIL) and the associated run time environment, Motif Resource Manager (MRM).

UIL is a descriptive language in which widget hierarchies can be defined statically. To this end, the display objects for the user interface (e.g. labels, buttons, menus or even Container widgets) are specified, together with the callbacks which are meaningful for them.

The following elements can be specified in a UIL module:

- Objects (i.e. widgets or gadgets) from which the user interface is made up
- Resources for these objects
- The object tree which describes the user interface
- Values to be read and evaluated by the application at run time.

The specification thus created is translated by a UIL compiler into a form which can be processed more rapidly. The resulting version is called the User Interface Definition (UID).

At run time, the Motif Resource Manager reads the UID files and builds up the corresponding widget hierarchy.

The principal advantage offered by the use of UIL is that it allows an application to be separated into form and function. The appearance of the user interface is not encoded in the application itself, but instead in a separate file (the UIL file). This makes it possible for the design of the interface and the development of the functionality of an application to be undertaken in parallel by separate development groups. It is then possible to make changes to the user interface without having to modify the actual source code.

UIL is of particular interest to the programmer creating an X/Motif application in C, because it permits the programming to be done at a "higher" level than in C.

However, for a IF/Prolog programmer using the OSF/Motif interface, this is hardly such a significant benefit. The IF/Prolog interface already uses a "higher-level" representation for many resource types, for example for colors or fonts, thus effectively eliminating the disadvantage of the static user interface (as in UIL).

Nevertheless, the IF/Prolog OSF/Motif Interface does include access to compiled UIL files. Subject to minor restrictions, the programmer can therefore use the same mechanisms as the C programmer. The UIL is accessed by predicates which work in a similar manner to the C functions.

The following restrictions apply:

- IF/Prolog predicates can only be linked to a fixed number of callbacks, which are specified in the UIL. Currently, 30 callbacks are supported.
User Interface Language UIL

- `mrmFetchWidgetOverride/5` only disables the widget name. In almost every case, the arguments can be modified by a subsequent explicit `xtSetValues/2`.

- At present, the values which can be passed from the IF/Prolog application to UIL identifiers are restricted to integers and strings.

- Exported integer tables which are fetched by the application must specify their length in the first element of the list. (e.g. `integer_table(4,1,2,3,4)`.)
UIL program example

module helloworld
    version = 'v1.0'
    names = case_sensitive

    procedure
        helloworld_button_activate();

    object
        helloworld_main : XmBulletinBoard widget {
            controls {
                XmLabel helloworld_label;
                XmPushButton helloworld_button;
            }
        }
    }

    object
        helloworld_button : XmPushButton widget {
            arguments {
                XmNx = 5;
                XmNy = 60;
                XmNlabelString = compound_string('Hello',separate=true)
                    &'World!';
            }
            callbacks {
                XmNactivateCallback = procedure
                    helloworld_button_activate();
            }
        }
    }

    object
        helloworld_label : XmLabel widget {
            arguments {
                XmNlabelString =
                    compound_string('Press button once',separate=true)
                    & compound_string('to change label;',separate=true)
                    & 'twice to exit.';
            }
        }

end module;
Use in an OSF/Motif program

:- import( motif).

program :-
  user_parameters(Parms),
  % Initialize the Xt Toolkit.
  % The "main" widget of the application must be the only child
  % of the widget.

  xtInitialize(toplevel, toplevel, 'helloworldclass',
    [['-uid', 'uidfile', sepArg]], ['HelloWorld' | Parms]),
  xtSetValues(toplevel, [ allowShellResize=true ]),

  % Obtain the file name for the UIL definition from a resource
  % file or the command line option -uid

  xtGetApplicationResources( toplevel,
    [[uidfile, 'uidfile', File, 'hello.uid']], []),

  mrmInitialize,
  % Define the MRM hierarchy (only one file)

  mrmOpenHierarchy([ File ], HierarchyId),
  % Register the callback routines so that the Resource Manager
  % can use them in creating the widget.

  mrmRegisterProcedureNames([ [ helloworld_button_activate,
      helloworld_button_activate, string ] ]),
  set_global(flag, false),

  % Call up MRM to create the push button and its Container
  % widget.

  mrmFetchWidget(HierarchyId, helloworld_main, toplevel,
    HelloWorld_main),

  % Initiate the management of the main window by the toplevel
  % widget (or by the widget provided for this purpose by UIL).
  % This instructs the system to realize the widget when the
  % toplevel widget is realized.

  xtManageChild(HelloWorld_main),
% Realize the toplevel widget. This has the effect that the
% widget hierarchy is mapped.

xtRealizeWidget(toplevel),

% Process the event loop.

xtMainLoop.

helloworld_button_activate(Widget, _ClientData, _CallbackData) :-
    xtSetValues( Widget, [ labelString='Goodbye\nWorld!' ] ),
    get_global(flag, Boolean),
    times_pressed(Boolean).

times_pressed(false):-
    write('Button pressed once'), nl,
    set_global(flag, true).

times_pressed(true) :-
    write('Button pressed twice'), nl,
    halt.
	times_pressed(_).
Chapter 3

Widgets and gadgets

In this chapter, the predefined widgets and gadgets that are available in IF/Prolog are described in alphabetical order. These comprise the Xt Intrinsics widget set plus the OSF/Motif widget set which supplements it. The hierarchy of these widgets is described in the section entitled Introduction to the X Window System and OSF/Motif.

More detailed information on programming with widgets is contained in the manuals on Xt Intrinsics ([16] and [17]) and OSF/Motif ([18] and [19]).

The predicates associated with the widgets are described in alphabetical order in the chapter entitled Reference section on predefined predicates (page 191). A summary of the predicates will be found in the chapter entitled Overview of predefined predicates by function (page 175).

Provided that IF/Prolog has been installed with the functionality of the OSF/Motif interface, the predefined OSF/Motif widgets will be available in IF/Prolog.

Predicates for creating widgets

The general format of the predicates is:

\[ \texttt{xmCreateWhat (?WidgetName, +ParentName, +ToolkitName, +ArgList)} \]

Depending on the specification for What, the corresponding widget type will be created. The resource values to be used can be specified in ArgList. Callbacks can only be provided by using the predicate \( \texttt{xtAddCallback/4} \). A valid name (atom) must be specified for the parent widget (ParentName). For WidgetName, either an atom (for a user-defined name) or a variable may be specified. If a variable is used for WidgetName, IF/Prolog will create and return a unique name.

ArgList is a list which contains resource specifications for the widget which is created. Further details of this will be found under the predicate \( \texttt{xtSetValues/2} \).

Format of the widget descriptions

The following descriptions of widgets and gadgets contain:

- the class, which must be specified when the widget is created
Widgets and gadgets

- the superclass(es) for the widget, whose resources it also receives
- the resources which are made available by the widget
- the callbacks which are processed by the widget
- the callback components which can be accessed
- a cross-reference to widgets which are used

The descriptions of the resources start with a table giving the following details:

- the resource name
- the data type of the resource value in IF/Prolog
- the default value for the resource
- the options for accessing the resource within the program.

The following abbreviations are used in the tables:

C (Create) The resource can only be set when the widget is created.
S (Set) The resource can be set if the widget already exists.
G (Get) The value of the resource can be interrogated in the program.

Following each table is a detailed description of each resource and the possible values which it may have.

As a basic rule, a widget class has all the resources which are available for its superclasses. The tables therefore only describe the resources which are used in addition to those of the appropriate superclasses.

In each case, all resources of the widget are itemized, including those which cannot be used from within IF/Prolog. For the latter resources, no corresponding IF/Prolog values are given.

Translators and accelerators

For the IF/Prolog OSF/Motif Interface, no changes have been made to the predefined translations and accelerators. They are therefore not described in this manual.
ApplicationShell

The ApplicationShell widget is generally the top widget in the widget tree for an application. It is automatically created when xtInitialize/5 is called.

Class: applicationShellWidgetClass
Superclasses: TopLevelShell, VendorShell, WMShell, Shell, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argc</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>argv</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

argc
Indicates the number of arguments in argv.
Value: cannot be used from within IF/Prolog

argv
Contains the argument list required by the session manager to start the application again if it has been aborted.
Value: cannot be used from within IF/Prolog

See also

DialogShell, TransientShell, TopLevelShell
ArrowButton (widget)

The ArrowButton widget is a rectangular button containing an arrow.

Class: xmArrowButtonWidgetClass
Superclasses: Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>C</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>armCallback</td>
<td>C</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>C</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>multiClick</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

activateCallback
Defines the callback list which will be called if the widget is activated.
Value: can only be set by xtAddCallback/4

armCallback
Defines the callback list which will be called if the widget is marked.
Value: can only be set by xtAddCallback/4

arrowDirection
Defines the direction in which the arrow mapped in the widget points.
Value: arrow_up, arrow_down, arrow_right, arrow_left

disarmCallback
Specifies the callback list which will be called if the marking of the widget is canceled.
Value: can only be set by xtAddCallback/4

multiClick
Specifies whether multiple clicking on the button in rapid succession will result in multiple execution of the function or not.
Value: multiclick_discard only the first click is processed
       multiclick_keep all clicks will be processed

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>activate</td>
<td>Button activated</td>
</tr>
<tr>
<td>armCallback</td>
<td>arm</td>
<td>Button marked</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>disarm</td>
<td>Button marking canceled</td>
</tr>
</tbody>
</table>

Motif Manual 38 IF/Prolog V5.2
### Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>click_count</td>
<td>integer</td>
<td>activate</td>
<td>Number of clicks</td>
</tr>
</tbody>
</table>
ArrowButton (gadget)

The ArrowButton gadget corresponds to the ArrowButton widget.

Class xmArrowButtonGadgetClass
Superclasses Gadget, RectObj, Object

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.
BulletinBoard

The BulletinBoard widget is a Container widget whose child widgets are dialog widgets. BulletinBoard provides its children with resources for dialog and geometry management.

Class: xmBulletinBoardWidgetClass
Superclasses: Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowOverlap</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>autoUnmanage</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>buttonFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>cancelButton</td>
<td>atom</td>
<td>SG</td>
<td>none</td>
</tr>
<tr>
<td>defaultButton</td>
<td>atom</td>
<td>SG</td>
<td>none</td>
</tr>
<tr>
<td>defaultPosition</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>dialogStyle</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>dialogTitle</td>
<td>atom, list</td>
<td>CSG</td>
<td>‘’</td>
</tr>
<tr>
<td>focusCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>labelFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>mapCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>noResize</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>resizePolicy</td>
<td>atom</td>
<td>CSG</td>
<td>resize_any</td>
</tr>
<tr>
<td>shadowType</td>
<td>atom</td>
<td>CSG</td>
<td>shadow_out</td>
</tr>
<tr>
<td>textFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>textTranslations</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>unmapCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
</tbody>
</table>

allowOverlap
Gives permission for child widgets to overlap.
Value: boolean values (true, false, on, off, yes, no)

autoUnmanage
Defines whether a widget automatically ceases to be managed when a button is activated.
Value: boolean values (true, false, on, off, yes, no)

buttonFontList
Defines which font is to be used for text items in the child widgets.
Value: a valid font list
cancelButton
  Specifies the widget id of the Cancel button.
  Value: the name of the widget

defaultButton
  Specifies the widget id of the default button. This is the button which can also be activated by the Return key.
  Value: the name of the widget

defaultPosition
  Defines whether the widget will be positioned beside its parent widget.
  Value: boolean values (true, false, on, off, yes, no)
dialogStyle
  inputMode resource of the vendor shell of the parent widget. That is, it specifies whether other widgets are prevented from accepting inputs from the user until the dialog for the widget is completed.

  Value:
  dialog_system_modal
    All inputs to other widgets, including those for different applications, are inhibited.
  dialog_full_application_modal
    Any dialog with widgets in the same application is inhibited.
  dialog_application_modal
    Inhibits inputs into ancestor widgets in the same application.
  dialog_modeless
    No other widgets are inhibited. This is the default setting if the parent widget is a dialog shell.
  dialog_work_area
    The default setting, if the parent widget is not a dialog shell.
dialogTitle
  Sets the title resource for the parent widget’s dialog shell.
  Value: a valid XmString
focusCallback
  Specifies the callback list which will be called if the widget is given the input focus.
  Value: can only be set by xtAddCallback/4
labelFontList
  Defines the font for children which are Label widgets.
  Value: a valid font list
mapCallback
Specifies the callback list which will be called if the widget is mapped.
Value: can only be set by xtAddCallback/4

marginHeight
Defines the distance, in pixels, between the top or bottom border of the widget and the child widget.
Value: an integer

marginWidth
Defines the distance, in pixels, between the left or right border of the widget and the child widget.
Value: an integer

noResize
Specifies whether mwm may use the size setting of the parent widget for the widget.
Values: boolean values (true, false, on, off, yes, no)

resizePolicy
Specifies the policy for setting the size of the BulletinBoard widget.
Value: resize_none size cannot be changed
        resize_any reduction or enlargement permitted
        resize_grow only enlargement permitted

shadowType
Defines the type of shadow for the BulletinBoard widget.
Value: shadow_in The widget appears to be sunk into the surface.
        shadow_out The widget appears raised relative to the surface.

The following two values use double shadows to create a three-dimensional impression.
        shadow_etched_in The widget appears to be sunk into the surface.
        shadow_etched_out The widget appears raised relative to the surface.

textFontList
Defines the fonts for children which are Text widgets.
Value: a valid font list

textTranslations
Defines the translation table for children which are Text widgets.
Value: a valid translation table
unmapCallback

Defines the callback list which is to be called when the widget is deactivated. This is only applicable if the parent widget is a DialogShell widget.

Value: can only be set by xtAddCallback/4

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>focusCallback</td>
<td>focus</td>
<td>Input focus accepted</td>
</tr>
<tr>
<td>mapCallback</td>
<td>map</td>
<td>Use the dialog shell of the parent widget</td>
</tr>
<tr>
<td>unmapCallback</td>
<td>unmap</td>
<td>Do not use the dialog shell of the parent widget</td>
</tr>
</tbody>
</table>
CascadeButton (widget)

The CascadeButton widget is a button which links together two menu areas or a menu bar and a menu area. If a CascadeButton widget is activated, an assigned cascade menu (pull-right menu) will be posted beneath or alongside the button.

Class  xmCascadeButtonWidgetClass
Superclasses  Label, Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>cascadePixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>cascadingCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>mappingDelay</td>
<td>integer</td>
<td>CSG</td>
<td>180</td>
</tr>
<tr>
<td>subMenuId</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
</tbody>
</table>

activateCallback

Defines the callback list which will be called if the widget is activated.

Value: can only be set by xtAddCallback/4

cascadePixmap

Defines the pixmap which will be displayed on the right of the menu to indicate that there is a cascade menu for the menu item.

Value: pixmap (name of a bitmap file)

cascadingCallback

Defines the callback list which will be called if the widget is activated, i.e. if the cascade menu is posted.

Value: can only be set by xtAddCallback/4

mappingDelay

Defines the time (in milliseconds) that is to elapse between marking of the cascade button and mapping of the widget.

Value: integer

subMenuId

Specifies the name of the associated pull-down menu.

Value: name of the widget
### Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>activate</td>
<td>Input focus accepted</td>
</tr>
<tr>
<td>cascadingCallback</td>
<td>cascading</td>
<td>Mapping of the cascade menu (called after menu is mapped)</td>
</tr>
</tbody>
</table>
CascadeButton (gadget)

The CascadeButton gadget corresponds to the CascadeButton widget.

Class \texttt{xmCascadeButtonGadgetClass}
Superclasses LabelGadget, Gadget, RectObj, Object

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.

See also

PushButton, DrawnButton, ToggleButton
Command

The Command widget is a composite widget used for command input. It consists of an input area and a list. Commands are entered in the input area and added to the list. After a command has been executed, it is transferred into a history list.

Class: xmCommandWidgetClass
Superclasses: SelectionBox, SelectionBox, BulletinBoard, Manager, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>atom, list</td>
<td>CSG</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>commandChangedCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>commandEnteredCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>historyItems</td>
<td>list</td>
<td>CSG</td>
<td>[ ]</td>
</tr>
<tr>
<td>historyItemCount</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>historyMaxItems</td>
<td>integer</td>
<td>CSG</td>
<td>100</td>
</tr>
<tr>
<td>historyVisibleItemCount</td>
<td>integer</td>
<td>CSG</td>
<td>8</td>
</tr>
<tr>
<td>promptString</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

command

Text which is output in the command input area.
Value: a valid XmString

commandChangedCallback

Defines the callback list which will be called if the text in the command line is changed.
Value: can only be set by xtAddCallback/4

commandEnteredCallback

Defines the callback list which will be is called if a command is input.
Value: can only be set by xtAddCallback/4

historyItems

Entries in the history list.
Value: list of XmStrings

historyItemCount

The number of history items entered. The value is set automatically to the length of the historyItems resource list.
Value: integer

historyMaxItems

Defines the maximum number of history items.
Value: integer
historyVisibleItemCount
  Defines the number of lines of the history list which are visible.
  Value: integer

promptString
  Specifies a character string which will be output as a prompt in the command
  line to request input from the user.
  Value: valid XmString

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandChangedCallback</td>
<td>command_changed</td>
<td>Command line text has been changed</td>
</tr>
<tr>
<td>commandEnteredCallback</td>
<td>command_entered</td>
<td>Command has been input</td>
</tr>
</tbody>
</table>

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valueString</td>
<td>list</td>
<td>all</td>
<td>Command (XmString)</td>
</tr>
</tbody>
</table>
**Composite**

The `compositeWidgetClass` class of widgets cannot be instantiated. It is only implemented as a common superclass for all Container widgets.

Class: `compositeWidgetClass`
Superclasses: `Core`

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>list</td>
<td>G</td>
<td>[]</td>
</tr>
<tr>
<td>insertPosition</td>
<td>-</td>
<td>CSG</td>
<td>-</td>
</tr>
<tr>
<td>numChildren</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
</tbody>
</table>

**children**

This is a list which contains all the children of the widget. It can only be read.

Value: list of widget names

**insertPosition**

Specifies the position at which a child widget is to be inserted into the list of children.

Value: cannot be used from within IF/Prolog

**numChildren**

Specifies the length of the list of child widgets.

Value: integer
Constraint

The constraintWidgetClass class of widgets cannot be instantiated. It is only implemented as a common superclass for all Container widgets. The special feature of the Constraint class is that it defines so-called constraint resources, which are not used by a widget from this class itself, but only by widgets which are its immediate children. The constraint resources contain information about size or position, including relative to other widgets.

The Constraint class does not define any resources of its own. These are specified in the appropriate subclasses of Constraint.

Class constraintWidgetClass
Superclasses Composite, Core
The coreWidgetClass class of widgets cannot be instantiated. It is only implemented as a common superclass for all widgets.

### Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>accelerators</td>
<td>atom</td>
<td>CSG</td>
<td></td>
</tr>
<tr>
<td>ancestorSensitive</td>
<td>atom</td>
<td>G</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>background</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>backgroundPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>borderColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>borderPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>borderWidth</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>colormap</td>
<td>-</td>
<td>CG</td>
<td></td>
</tr>
<tr>
<td>depth</td>
<td>integer</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>destroyCallback</td>
<td>-</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>initialResourcePersistant</td>
<td>atom</td>
<td>CG</td>
<td>true</td>
</tr>
<tr>
<td>mappedWhenManaged</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>screen</td>
<td>-</td>
<td>CG</td>
<td></td>
</tr>
<tr>
<td>sensitive</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>translations</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>width</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>x</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
</tbody>
</table>

- **accelerators**: Specifies the accelerator table which will be used for the widget.
  
  Value: valid accelerator table

- **ancestorSensitive**: Specifies whether the parent widget receives the input events.
  
  Value: boolean values (true, false, on, off, yes, no)

- **background**: Defines the background color of the window.
  
  Value: name of a valid color

- **backgroundPixmap**: Specifies the pattern to be used for the window background.
  
  Value: pixmap (name of a bitmap file)
borderColor
  Defines the color of the window border.
  Value: name of a valid color.

borderPixmap
  Defines the pattern for the window border.
  Value: pixmap (name of a bitmap file)

borderWidth
  Defines the width of the window border in pixels.
  Value: integer

colormap
  Specifies the color table to be used for the widget.
  Value: cannot be used from within IF/Prolog

depth
  Defines the number of color levels for the window, i.e. the number of bits used for the display of each pixel.
  Value: integer

destroyCallback
  Specifies which callback list is to be called when the widget is destroyed.
  Value: can only be set by xtAddCallback/4

height
  Defines the height of the window in pixels.
  Value: integer

initialResourcesPersistant
  Specifies whether the resources for the widget are to be referenced. If the value is false, the resources will be referenced, so that it is possible for the memory used for these resources to be released if the widget is destroyed. This is only relevant if there is a possibility of the widget being destroyed before the application terminates. If not, this resource should remain set to true.
  Value: boolean values (true, false, on, off, yes, no)

mappedWhenManaged
  Defines whether the widget is also to be mapped after it has been realized and passed to the parent widget for management.
  Value: boolean values (true, false, on, off, yes, no)

screen
  Specifies the screen on which the widget is to be mapped.
  Value: cannot be used from within IF/Prolog
Widgets

sensitive
Specifies whether the widget will accept input events.
Value: boolean values (true, false, on, off, yes, no)

translations
Specifies the translation table to be used for the widget.
Value: valid translation table

width
Defines the width of the window in pixels.
Value: integer

x
Specifies the X coordinate of the upper left corner of the widget, relative to its parent widget.
Value: integer

y
Specifies the Y coordinate of the upper left corner of the widget, relative to its parent widget.
Value: integer

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destroyCallback</td>
<td>-</td>
<td>Widget is destroyed</td>
</tr>
</tbody>
</table>
DialogShell

The DialogShell class of widgets comprises Shell widgets which can be used with a dialog widget as child widget.

Class xmDialogShellWidgetClass
Superclasses VendorShell, WMShell, Shell, Composite, Core

The xmDialogShellWidgetClass class does not define any new resources, but merely overwrites the value of the deleteResponse resource in class vendorShellClass with the value unmap.

See also

ApplicationShell, TransientShell, TopLevelShell
Display

The Display object is used to store information specific to a display. It is created automatically when the user calls xtInitialize/5/6. Initial values for the resources can only be set in resource files.

Class                     xmDisplayWidgetClass
Superclasses              ApplicationShell, TopLevelShell, VendorShell, WMShell, Shell, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultVirtualBindings</td>
<td>atom</td>
<td>CG</td>
<td>-</td>
</tr>
<tr>
<td>dragInitiatorProtocolStyles</td>
<td>atom</td>
<td>CG</td>
<td>-</td>
</tr>
<tr>
<td>dragReceiverProtocolStyle</td>
<td>atom</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

defaultVirtualBindings             
   Specifies the default virtual bindings for the display.

dragInitiatorProtocolStyles        
   Specifies drag and drop protocol requirements for initiator client.
   Value: drag_preregister  do not use the dynamic protocol
drag_dynamic                      do not use preregistered drop site information
drag_none                         disable drag and drop (for this client)
drag_drop_only                    support just dragging
drag_prefer_dynamic               support dynamic and preregistered protocol, but prefer dynamic
drag_prefer_preregister           support dynamic and preregistered protocol, but prefer preregistered
drag_prefer_receiver              support dynamic and preregistered protocol, but defer to preference of receiver

dragReceiverProtocolStyle          
   Specifies drag and drop protocol requirements for receiver client.
   Value: drag_preregister  do not use the dynamic protocol
drag_dynamic                      do not use preregistered drop site information
drag_none                         disable drag and drop (for this client)
drag_drop_only                    support just dragging
drag_prefer_dynamic               support dynamic and preregistered protocol, but prefer dynamic
drag_prefer_preregister           support dynamic and preregistered protocol, but prefer preregistered
Compatibility

Display is not available in the interface to OSF/Motif 1.1.

See also

Screen
**DragContext**

The DragContext widget is a special widget used in drag and drop transactions. It is implicitly created when calling the predicate `xmDragStart/3`.

**Class**  xmDragContextClass  
**Superclasses**  Core

### Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>blendModel</td>
<td>atom</td>
<td>CG</td>
<td>blend_all</td>
</tr>
<tr>
<td>clientData</td>
<td>atom</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>convertProc</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>cursorBackground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>cursorForeground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>dragDropFinishCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>dragMotionCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>dragOperations</td>
<td>list</td>
<td>C</td>
<td>[drop_copy, drop_move]</td>
</tr>
<tr>
<td>dropFinishCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>dropSiteEnterCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>dropSiteLeaveCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>dropStartCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>exportTargets</td>
<td>list</td>
<td>CSG</td>
<td>[]</td>
</tr>
<tr>
<td>incremental</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>invalidCursorForeground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>noneCursorForeground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>numExportTargets</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>operationChangedCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>operationCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>sourceCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>sourcePixmapIcon</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>stateCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>topLevelEnterCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>topLevelLeaveCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>validCursorForeground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

**blendModel**

Specifies a combination of DragIcons

Value:  
- blend_all: blend source, state, an operation icons  
- blend_state_source: blend state and source icons only  
- blend_just_source: use just source icon  
- blend_one: use no drag-over visual

**clientData**

Data passed to `convertProc`.  

Motif Manual  58  IF/Prolog V5.2
convertProc
Predicate used to convert the source data to the format(s) requested by the user. convertProc must be a user-defined predicate with four parameters:
convert Predicate( Widget, SelectionId, ReturnId, ReturnValue)
The SelectionId and ReturnId are Prolog atoms that are automatically interned as OSF/Motif atoms. ReturnId and ReturnValue are output parameters which must be bound to an atom or a term respectively on return from the predicate.

cursorBackground
Specifies the cursor background.
Value: atom

cursorForeground
Specifies the cursor foreground.
Value: atom

dragDropFinishCallback
Specifies the list of callbacks that are called when the transaction is completed.
Value: can only be set by xtAddCallback/4

dragMotionCallback
Specifies the list of callbacks that are called when the cursor moves.
Value: can only be set by xtAddCallback/4

dragOperations
Specifies a set of valid operations.
Value: drop_copy
drop_link
drop_move
drop_noop

dropFinishCallback
Specifies the list of callbacks that are called when the drop is completed.
Value: can only be set by xtAddCallback/4

dropSiteEnterCallback
Specifies the list of callbacks that are called when the pointer enters a drop site.
Value: can only be set by xtAddCallback/4
Widgets

DragContext

dropSiteLeaveCallback
    Specifies the list of callbacks that are called when the pointer leaves a drop site.
    Value: can only be set by xtAddCallback/4

dropStartCallback
    Specifies the list of callbacks that are called when a drop is initiated.
    Value: can only be set by xtAddCallback/4

exportTargets
    Specifies the list of targets that this site can convert data to.
    Value: List of atoms.

incremental
    Specifies whether the incremental selection transfer mechanism can be used.
    Value: boolean values (true, false, on, off, yes, no)

invalidCursorForeground
    Specifies the cursor pixel value used when the state is invalid.
    Value: atom

noneCursorForeground
    Specifies the cursor pixel value used when the state is none.
    Value: atom

numExportTargets
    Specifies the number of entries in the export target list.
    Value: integer

operationChangedCallback
    Specifies the list of callbacks that are called when a drag is started and the user requests a different operation be applied to the drop.
    Value: can only be set by xtAddCallback/4

operationCursorIcon
    Specifies the cursor icon used to designate the type of operation.
    Value: atom

sourceCursorIcon
    Specifies the cursor icon used to represent the source.
    Value: atom

sourcePixmapIcon
    Specifies the pixmap icon used to represent the source.
    Value: atom
stateCursorIcon

Specifies the pixmap icon used to designate the state of a drop site.

Value: atom

topLevelEnterCallback

Specifies the list of callbacks that are called when the cursor enters a top level window.

Value: can only be set by xtAddCallback/4

topLevelLeaveCallback

Specifies the list of callbacks that are called when the cursor leaves a top level window.

Value: can only be set by xtAddCallback/4

validCursorForeground

Specifies the foreground pixel value for a valid cursor icon.

Value: atom

### Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dragDropFinishCallback</td>
<td>drag_drop_finish</td>
<td>Transaction finished</td>
</tr>
<tr>
<td>dragMotionCallback</td>
<td>drag_motion</td>
<td>Drag cursor moved</td>
</tr>
<tr>
<td>dropFinishCallback</td>
<td>drop_finish</td>
<td>Drop finished</td>
</tr>
<tr>
<td>dropSiteEnterCallback</td>
<td>drop_site_enter</td>
<td>Drop site entered</td>
</tr>
<tr>
<td>dropSiteLeaveCallback</td>
<td>drop_site_leave</td>
<td>Drop site left</td>
</tr>
<tr>
<td>dropStartCallback</td>
<td>drop_start</td>
<td>Drop started</td>
</tr>
<tr>
<td>operationChangedCallback</td>
<td>operation_changed</td>
<td>Operation changed</td>
</tr>
<tr>
<td>topLevelEnterCallback</td>
<td>top_level_enter</td>
<td>Top level window entered</td>
</tr>
<tr>
<td>topLevelLeaveCallback</td>
<td>top_level_leave</td>
<td>Top level window left</td>
</tr>
</tbody>
</table>

### Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>atom</td>
<td>*all</td>
<td>Reason for callback</td>
</tr>
<tr>
<td>timeStamp</td>
<td>integer</td>
<td>*all</td>
<td>Time of event</td>
</tr>
<tr>
<td>operation</td>
<td>atom</td>
<td>*</td>
<td>Drop site operation</td>
</tr>
<tr>
<td>operations</td>
<td>list</td>
<td>*</td>
<td>List of supported operations</td>
</tr>
<tr>
<td>dragProtocolStyle</td>
<td>atom</td>
<td>*</td>
<td>Protocol style adopted by initiator</td>
</tr>
<tr>
<td>dropAction</td>
<td>atom</td>
<td>*</td>
<td>Identifies the drop action</td>
</tr>
<tr>
<td>dropSiteStatus</td>
<td>atom</td>
<td>*</td>
<td>Whether or not drop site is valid</td>
</tr>
<tr>
<td>x</td>
<td>integer</td>
<td>*</td>
<td>x coordinate of the pointer</td>
</tr>
<tr>
<td>y</td>
<td>integer</td>
<td>*</td>
<td>Y coordinate of the pointer</td>
</tr>
</tbody>
</table>

*: Please consult OSF/Motif documentation on callback components and their values.
Compatibility

DragContext is not available in the interface to OSF/Motif 1.1.

See also

DragIcon, DropSite
**DragIcon**

The DragIcon object is used as a visual representation of the source data during a drag transaction.

Class: xmDragIconObjectClass  
Superclasses: Object

### Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>atom</td>
<td>CSG</td>
<td>attachment_north_west</td>
</tr>
<tr>
<td>depth</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>height</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>hotX</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>hotY</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>mask</td>
<td>atom</td>
<td>CSG</td>
<td>unspecified_pixmap</td>
</tr>
<tr>
<td>offsetX</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>offsetY</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>pixmap</td>
<td>atom</td>
<td>CSG</td>
<td>unspecified_pixmap</td>
</tr>
<tr>
<td>width</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
</tbody>
</table>

**attachment**  
Specifies the relative attachments of the state or operation icons to the source icon.

Value:
- attach_north_west
- attach_north
- attach_east
- attach_south_east
- attach_south
- attach_south_west
- attach_west
- attach_center
- attach_hot
- attach_north_east

**depth**  
Specifies the depth of the pixmap.

Value: integer

**height**  
Specifies the height of the pixmap.

Value: integer

**hotX**  
Specifies the x-coordinate of the hotspot of the DragIcon relative to the origin of the pixmap used.

Value: integer
<table>
<thead>
<tr>
<th>DragIcon</th>
<th>DragIcon</th>
</tr>
</thead>
</table>
| **hotY**               | Specifies the y-coordinate of the hotspot of the DragIcon relative to the origin of the pixmap used.  
Value: integer          |
| **mask**               | Specifies the pixmap used as the DragIcon mask pixmap.  
Value: atom              |
| **offsetX**            | Specifies the positions of the state or operation icons relative to the source icon.  
Value: integer          |
| **offsetY**            | Specifies the positions of the state or operation icons relative to the source icon.  
Value: integer          |
| ** pixmap**            | Specifies the pixmap used as the DragIcon pixmap.  
Value: atom              |
| **width**              | Specifies the width of the pixmap.  
Value: integer          |

**Compatibility**

DragIcon is not available in the interface to OSF/Motif 1.1.

**See also**

[DragContext], [DropSite], [DropTransfer]
Draw

The Draw widget is an empty widget for presentation of graphics; it has a series of external functions which give the user of the widget access to the basic drawing functions of Xlib. It manages graphic resources, and automatically repaints the contents of the screen. All the external functions are available as IF/Prolog predicates.

A comprehensive description of the Draw widget will be found in the chapter entitled *The Draw widget* (page 169).

Class: `xmDrawWidgetClass`

Superclasses: `Manager`, `Constraint`, `Composite`, `Core`

### Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>asciiCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>bitmap0</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>bitmap9</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>buttonCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>clearBeforeRepaint</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>color0</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>color9</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>compressMotion</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>fixedRefreshPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>flushAfterDrawing</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>font</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>font0</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>font9</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>fontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>motionCallback</td>
<td>atom</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>readDumpToPixmap</td>
<td>atom</td>
<td>CG</td>
<td>true</td>
</tr>
<tr>
<td>refreshFromPixmap</td>
<td>atom</td>
<td>CG</td>
<td>true</td>
</tr>
<tr>
<td>repaintBeforeDrawingIfExposed</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>repaintGrownSize</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>saveOpList</td>
<td>atom</td>
<td>CG</td>
<td>false</td>
</tr>
<tr>
<td>savingsOps</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
</tbody>
</table>
### WritingPixmap

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>writingPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
</tbody>
</table>

**asciiCallback**

This is a callback list whose callbacks are called by the action asciiCallback(). The action should be initiated by a key event. In this case, the call data for the callback is draw_ascii.

Value: in IF/Prolog, can only be set by xtAddCallback/4.

**bitmap0 ... bitmap9**

Specify bitmaps to be set as the pattern for a graphics context grid. They can also be used with the predicate xtDrawCopyBitmap/4.

Value: pixmap (name of a bitmap file)

**buttonCallback**

This is a callback list whose callbacks are called by the action buttonCallback(). The action should be initiated by a button event. In this case, the call data for the callback is draw_button.

Value: in IF/Prolog, can only be set by xtAddCallback/4.

**clearBeforeRepaint**

Specifies whether or not the contents of the window will be cleared before the screen is repainted. It is better not to clear the window, but it may be necessary to set clearBeforeRepaint to true if the copy area or rubber-banding mode functions are used.

Value: boolean values (true, false, on, off, yes, no)

**color0 ... color9**

Specify the color values which will be set for the pixels for the foreground or the background of the graphics context, or for the grid. color0 and color1 are taken from the values of the foreground and background resources when these are initialized.

Value: valid color

**compressMotion**

Specifies whether or not the motion events will be compressed. Compression is set by default in order to improve performance; however, it can be switched off if the pointer movements are to be more closely tracked.

Value: boolean values (true, false, on, off, yes, no)

**fixedRefreshPixmap**

Specifies whether or not the pixmap that controls refreshing of the widget is to be fixed in size based on the widget when it was created. This can be used to improve performance for applications which have a layout with a fixed maximum size.

Value: boolean values (true, false, on, off, yes, no)
flushAfterDrawing
Specifications whether or not the output buffer for the window is flushed after every operation. This is normally unnecessary, since the buffer will automatically be flushed in the main loop of the program (MainLoop). If immediate drawing is required, this resource can be set to true.
Value: boolean values (true, false, on, off, yes, no)

foreground
 Specifies the color of the foreground for the graphics context, and is set at the start to the appropriate value. This color is also assigned to the color1 resource during initialization.
Value: valid color

font
 Specifies the font to be used for text items in the graphics context, and will be set to the appropriate value at the start. The value of this resource is also assigned to the font0 resource during initialization.
Value: name of a font

font0 ... font9
Specify the fonts for text items in the graphics context. font0 is copied from the font resource during initialization.
Value: name of a font

motionCallback
This is a callback list whose callbacks are called by the action motionCallback(). The action should be initiated by a motion event. In this case, the call data for the callback is draw.motion.
Value: can only be set by xtAddCallback/4

repaintBeforeDrawingIfExposed
Specifies whether or not expose events will be notified before a draw operation is executed. It is useful to set this resource to true if no entry is made into the main event loop for a long time because a large number of drawings has to be prepared.
Value: boolean values (true, false, on, off, yes, no)

The Draw widget can save its drawings either as a pixmap or as a list containing the operations, from which it can recall them. The following resources specify which mechanism is used. An additional resource specifies how the memory dump for a window is to be read.

refreshFromPixmap
Specifies whether or not all the drawing operations will be executed both in the window and in an internal pixmap. The default setting for this resource is true. Repainting of the contents of the window is then carried
out using the pixmap. Only the predicate `xtDrawRepaint/1` uses the list of operations, if this has been saved. If the resource is set to `false`, repainting is performed using the operations list. This resource can only be set when the widget is created.

Value: boolean values (true, false, on, off, yes, no)

**repaintGrownSize**

Specifies whether the contents of a window which has been enlarged are to be repainted using the operations list. This enables drawings to be repainted to extend beyond their old borders.

This resource only has an effect if the resources `refreshFromPixmap` and `saveOpList` have the value `true`.

Value: boolean values (true, false, on, off, yes, no)

**saveOpList**

Specifies whether or not all the operations which are undertaken using the Draw widget are to be saved in a list, from which the contents of the window can be repainted. This will apply if the `refreshFromPixmap` resource has the value `false`, or if the window has been enlarged and `repaintGrownSize` has the value `true`. The operations in the list can be explicitly updated using `xtDrawRepaint/1`.

Value: boolean values (true, false, on, off, yes, no)

**savingsOps**

Specifies whether or not saving of operations in the list is to be suspended.

Value: boolean values (true, false, on, off, yes, no)

**writingPixmap**

Specifies whether or not recording of the drawing in the pixmap for repainting purposes is to be suspended.

Value: boolean values (true, false, on, off, yes, no)

**readDumpToPixmap**

Specifies whether or not the predicate `xtDrawReadWindowDump/8` is to save the image which it is reading as a pixmap. The image can be copied from this pixmap when the window is repainted. If the value `false` is assigned to this resource, then a new call will be made to `xtDrawReadWindowDump/8` each time that a repaint is carried out.

Value: boolean values (true, false, on, off, yes, no)

<table>
<thead>
<tr>
<th>CallbackName</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asciiCallback</td>
<td>draw_ascii</td>
<td>Key pressed</td>
</tr>
<tr>
<td>buttonCallback</td>
<td>draw_button</td>
<td>Button pressed</td>
</tr>
<tr>
<td>motionCallback</td>
<td>draw_motion</td>
<td>Button marking canceled</td>
</tr>
</tbody>
</table>

Callbacks
Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>integer</td>
<td>draw_ascii</td>
<td>Key code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>draw_button</td>
<td>Button number</td>
</tr>
</tbody>
</table>

Actions of the Draw widget

asciiCallback()

Calls the callbacks listed in the asciiCallback list, using the keycode as the call value, if the initiating event is a key event. Otherwise the call value will be -1.

buttonCallback()

Calls up the callbacks listed in the buttonCallback list, using the button number as the call value, if the initiating event is a button event. Otherwise the call value will be -1.

motionCallback()

Calls up the callbacks listed in the motionCallback list, with the call value 0, provided that the initiating event is a motion event. Otherwise the call value will be -1.

In addition, if the motionCallback() action was initiated by a motion event, it checks whether compression of the motion event is required or not. If it is, it calls XSync(); this waits for all events which have not yet been processed, and checks whether these include further motion events. All these events, except the last, are discarded. The pointer position for the last event will be saved.

clear()

Clears the contents of the Draw widget.

If any of the actions are initiated by an event which includes information about the position of the pointer, its coordinates will be stored so that the application program can later refer back to it using xtDrawGetXY/3.

Default translations

"<BtnDown> buttonCallback() \n<KeyPress>: asciiCallback()";

See also

DrawingArea
**DrawingArea**

The DrawingArea widget is an empty widget whose sole function is to call various callbacks. It is not particularly suitable for use from within IF/Prolog, because direct Xlib calls cannot be made from within IF/Prolog. The Draw widget can be used instead.

Class: xmDrawingAreaWidgetClass  
Superclasses: Manager, Constraint, Composite, Core

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>exposeCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>inputCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>resizeCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>resizePolicy</td>
<td>atom</td>
<td>CSG</td>
<td>resize_any</td>
</tr>
</tbody>
</table>

**exposeCallback**  
Specifies the callback list to be called when an expose event occurs.  
Value: can only be set by xtAddCallback/4

**inputCallback**  
Specifies the callback list to be called when an input has been made using the mouse or keyboard.  
Value: can only be set by xtAddCallback/4

**marginHeight**  
Defines the distance, in pixels, between the top or bottom corner of the widget and the child widget.  
Value: integer

**marginWidth**  
Defines the distance, in pixels, between the left or right corner of the widget and the child widget.  
Value: integer

**resizeCallback**  
Specifies the callback list to be called if a change is made to the size of the widget.  
Value: can only be set by xtAddCallback/4
DrawingArea Widgets

resizePolicy

Specifies the policy for setting the size of the DrawingArea widget.

Value:  resize_none size cannot be changed
        resize_any reduction or enlargement permitted
        resize_grow only enlargement permitted

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exposeCallback</td>
<td>expose</td>
<td>Expose event has occurred</td>
</tr>
<tr>
<td>inputCallback</td>
<td>input</td>
<td>Key or mouse event has occurred</td>
</tr>
<tr>
<td>resizeCallback</td>
<td>resize</td>
<td>Resize event has occurred</td>
</tr>
</tbody>
</table>

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>opaque</td>
<td>all</td>
<td>X Window</td>
</tr>
</tbody>
</table>

See also

Draw
DrawnButton

The DrawnButton widget behaves in a similar way to the PushButton widget. In addition to the output of a label or a pixmap, a graphic defined by the application can also be output. It is not suitable for use from within IF/Prolog, because direct Xlib calls from within IF/Prolog are not possible. The Draw widget can be used instead.

Class xmDrawnButtonWidgetClass
Superclasses Label, Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>armCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>exposeCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>multiClick</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>pushButtonEnabled</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>resizeCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>shadowType</td>
<td>atom</td>
<td>CSG</td>
<td>shadow_etched_in</td>
</tr>
</tbody>
</table>

activateCallback
Defines the callback list which will be called if the button is activated.
Value: can only be set by xtAddCallback/4

armCallback
Defines the callback list which will be called if the button is marked.
Value: can only be set by xtAddCallback/4

disarmCallback
Defines the callback list which will be called if the button marking is canceled.
Value: can only be set by xtAddCallback/4

exposeCallback
Defines the callback list which will be called if an expose event occurs.
Value: can only be set by xtAddCallback/4

multiClick
Specifies whether multiple clicking on the button in rapid succession will result in multiple execution of the function or not.
Value: multiclick_discard only the first click is processed
     multiclick_keep all clicks will be processed
Debeja

pushButtonEnabled

Specifies whether or not a three-dimensional shadow will be mapped around the button.

Value: boolean values (true, false, on, off, yes, no)

shadowType

Defines the type of shadow.

Value:
- shadow_in: The widget appears to be sunk into the surface.
- shadow_out: The widget appears raised relative to the surface.
- The following two values use double shadows to create a three-dimensional impression:
  - shadow_etched_in: The widget appears to be sunk into the surface.
  - shadow_etched_out: The widget appears raised relative to the surface.

resizeCallback

Specifies the callback list to be called if a resize event occurs.

Value: can only be set by xtAddCallback/4

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>activate</td>
<td>Button is activated</td>
</tr>
<tr>
<td>armCallback</td>
<td>arm</td>
<td>Button is marked</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>disarm</td>
<td>Marking is canceled</td>
</tr>
<tr>
<td>exposeCallback</td>
<td>expose</td>
<td>Expose event has occurred</td>
</tr>
<tr>
<td>resizeCallback</td>
<td>resize</td>
<td>Resize event has occurred</td>
</tr>
</tbody>
</table>

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>opaque</td>
<td>all</td>
<td>X Window</td>
</tr>
</tbody>
</table>

See also

PushButton, ToggleButton, CascadeButton
Widgets

**DropSite**

The DropSite is used to hold drag and drop data for a widget or gadget registered as a drop site. Use `xmDropSiteRegister/2` to register a widget as a drop site, `xmDropSiteUpdate/2` to set resource values, and `xmDropSiteRetrieve/2` to access resource values.

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>animationMask</td>
<td>atom</td>
<td>CG</td>
<td>unspecified_pixmap</td>
</tr>
<tr>
<td>animationPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>unspecified_pixmap</td>
</tr>
<tr>
<td>animationPixmapDepth</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>animationStyle</td>
<td>atom</td>
<td>CSG</td>
<td>drag_under_highlight</td>
</tr>
<tr>
<td>dragProc</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>dropProc</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>dropRectangles</td>
<td>list</td>
<td>CSG</td>
<td>[[X, Y, Width, Height]]</td>
</tr>
<tr>
<td>dropSiteActivity</td>
<td>atom</td>
<td>CSG</td>
<td>drop_site_active</td>
</tr>
<tr>
<td>dropSiteOperations</td>
<td>list</td>
<td>CSG</td>
<td>[drop_move, drop_copy]</td>
</tr>
<tr>
<td>dropSiteType</td>
<td>atom</td>
<td>CG</td>
<td>drop_site_simple</td>
</tr>
<tr>
<td>importTargets</td>
<td>list</td>
<td>CSG</td>
<td>[]</td>
</tr>
<tr>
<td>numDropRectangles</td>
<td>integer</td>
<td>G</td>
<td>1</td>
</tr>
<tr>
<td>numImportTargets</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
</tbody>
</table>

**animationMask**

Specifies the mask to use with the animation pixmap.

Value: atom

**animationPixmap**

Specifies the pixmap for drag-under animation.

Value: atom

**animationPixmapDepth**

Specifies the depth of the animation pixmap.

Value: integer

**animationStyle**

Specifies the drag-under animation style.

Value: 
- drag_under_highlight
- drag_under_shadow_out
- drag_under_shadow_in
- drag_under_pixmap
- drag_under_none

**dragProc**

Specifies the procedure that is invoked when the drop site receives a crossing, motion, or operation changed message.

Value: atom, name of a user defined callback predicate.
### DropSite Widgets

**dropProc**
- Specifies the procedure that is invoked when a drop occurs.
- Value: atom, name of a user defined callback predicate.

**dropRectangles**
- Specifies a list of rectangles that describe the shape of a drop site.
- Value: list of list, having the members x, y, width, and height.

**dropSiteActivity**
- Indicates whether a drop site is active or inactive.
- Value: atom, drop_site_active or drop_site_inactive.

**dropSiteOperations**
- Specifies the set of valid operations associated with a drop site.
- Value: atom, drop_copy, drop_link, drop_move, or drop_noop.

**dropSiteType**
- Specifies the type of the drop site.
- Value: drop_site_simple (No children of the widget are registered as drop sites.)
- drop_site_composite (The widget will have children that are registered as drop sites.)

**importTargets**
- Specifies the list of target atoms that this drop site accepts. The atoms are implicitly interned in IF/Prolog.
- Value: atom

**numDropRectangles**
- Specifies the number of drop rectangles.
- Value: integer

**numImportTargets**
- Specifies the number of import targets.
- Value: integer

### Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropProc</td>
<td>drop_site_enter_message</td>
<td>Enter a drop site</td>
</tr>
<tr>
<td>dropProc</td>
<td>drop_site_leave_message</td>
<td>Leave a drop site</td>
</tr>
<tr>
<td>dropProc</td>
<td>drag_motion_message</td>
<td>Drag cursor moved</td>
</tr>
<tr>
<td>dropProc</td>
<td>operation_changed_message</td>
<td>Operation changed</td>
</tr>
<tr>
<td>dropProc</td>
<td>drop_message</td>
<td>Drop occurred</td>
</tr>
</tbody>
</table>

**IF/Prolog V5.2**
Widgets

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>atom</td>
<td>all</td>
<td>Reason for callback</td>
</tr>
<tr>
<td>timeStep</td>
<td>integer</td>
<td>all</td>
<td>Time of event</td>
</tr>
<tr>
<td>dragContext</td>
<td>atom</td>
<td>all</td>
<td>Drop site operation</td>
</tr>
<tr>
<td>x</td>
<td>integer</td>
<td>all</td>
<td>x coordinate of the pointer</td>
</tr>
<tr>
<td>y</td>
<td>integer</td>
<td>all</td>
<td>y coordinate of the pointer</td>
</tr>
<tr>
<td>dropSiteStatus</td>
<td>atom</td>
<td>all</td>
<td>Whether drop site is valid</td>
</tr>
<tr>
<td>operation</td>
<td>atom</td>
<td>all</td>
<td>List of supported operations</td>
</tr>
<tr>
<td>operations</td>
<td>list</td>
<td>all</td>
<td>List of supported operations</td>
</tr>
<tr>
<td>dropAction</td>
<td>atom</td>
<td>all</td>
<td>Identifies the drop action</td>
</tr>
<tr>
<td>animate</td>
<td>atom</td>
<td>all but drop_message</td>
<td>Whether drag-under effects are supplied</td>
</tr>
</tbody>
</table>

Compatibility

DropSite is not available in the interface to OSF/Motif 1.1.

See also

DragIcon, DropContext
FileSelectionBox

The FileSelectionBox widget is a composite widget for searching for files within a file hierarchy. A list of files and directories is displayed; this shows file and directory names that contain a pattern of characters which is specified in a mask.

Class xmFileSelectionBoxWidgetClass
Superclasses SelectionBox, BulletinBoard, Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>directoryValid</td>
<td>atom</td>
<td>SG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>dirListItems</td>
<td>atom, list</td>
<td>SG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>dirListItemCount</td>
<td>integer</td>
<td>G</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>dirListLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Directories'</td>
</tr>
<tr>
<td>dirMask</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>dirSearchProc</td>
<td>-</td>
<td>CSG</td>
<td>-</td>
</tr>
<tr>
<td>dirSpec</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>fileListItems</td>
<td>list</td>
<td>SG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>fileListItemCount</td>
<td>integer</td>
<td>G</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>fileListLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Files'</td>
</tr>
<tr>
<td>fileSearchProc</td>
<td>-</td>
<td>CSG</td>
<td>-</td>
</tr>
<tr>
<td>fileTypeMask</td>
<td>atom</td>
<td>CSG</td>
<td>file_regular</td>
</tr>
<tr>
<td>filterLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Filter'</td>
</tr>
<tr>
<td>listUpdated</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>noMatchString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'[]'</td>
</tr>
<tr>
<td>pattern</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>qualifySearchDataProc</td>
<td>-</td>
<td>CSG</td>
<td>-</td>
</tr>
</tbody>
</table>

directory

Specifies the directory to be used as the starting point for the search in the file tree. If this value is NULL or empty, the search will start in the current directory.

Value: valid XmString

directoryValid

This will be set by the procedure which carries out the search. The value will be true if the directory specified to the procedure can be searched; the value will be false if the procedure was not called.

Value: boolean values (true, false, on, off, yes, no)

dirListItems

Defines the entries in the directory list.

Value: list of XmStrings
Widgets

FileSelectionBox

dirListItemCount
  Defines the number of entries in the directory list. The value is set automatically to the length of the dirListItems resource list.
  Value: integer

dirListLabelString
  Defines the character string which is to be output as the label for the directory list.
  Value: valid XmString

dirMask
  Specifies the mask for the directories which are to be output.
  Value: valid XmString

dirSearchProc
  Specifies a procedure to be executed in order to find a directory within a file tree.
  Value: cannot be used from within IF/Prolog

dirSpec
  Defines the full pathname of the file.
  Value: valid XmString

fileListItems
  Defines the entries in the file list.
  Value: list of XmStrings

fileListItemCount
  Defines the number of entries in the file list. The value is set automatically to the length of the fileListItems resource list.
  Value: integer

fileListLabelString
  Defines the character string which is to be output as the label for the file list.
  Value: valid XmString

fileSearchProc
  The name of the procedure to be executed in order to find a file.
  Value: cannot be used from within IF/Prolog

fileTypeMask
  Specifies the type of files to be output in the file list.
  Value: file_regular ordinary files only
         file_directory directories only
         file_any_type files and directories

Motif Manual 78 IF/Prolog V5.2
filterLabelString
Text to be output in the label above the directory mask.
Value: valid XmString

listUpdated
This will be given a value by the procedures which carry out the file or directory search. If the value is true, the list of directories or files has been updated.
Value: boolean values (true, false, on, off, yes, no)

noMatchString
Defines a character string, to be output in the first line of the file list if the list is empty, i.e. if no matching file has been found.
Value: valid XmString

pattern
Defines the search pattern which will be used together with the value of the directory resource to specify the files and directories that are to be output.
Value: valid XmString

qualifySearchDataProc
Specifies a procedure which will be called to edit the user’s inputs for use by the procedures which perform the file and directory searches. This will be used for the directory mask, the initial directory and the search pattern.
Value: cannot be used from within IF/Prolog

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valueString</td>
<td>list</td>
<td>all</td>
<td>Current value of dirSpec</td>
</tr>
<tr>
<td>dir</td>
<td>list</td>
<td>all</td>
<td>Current value of directory</td>
</tr>
<tr>
<td>pattern</td>
<td>list</td>
<td>all</td>
<td>Current value of pattern</td>
</tr>
</tbody>
</table>
Form

The Form widget is a Container widget which provides its child widgets with functions for managing their position and size. The position and size of the children are defined in relation to the position and size of the Form widget. This relationship has an effect when the size of the Form widget is altered, new children are added, the size of one of its children is altered, children are removed. For any of the four sides of a child widget, rules can be specified to define its behavior when there is a change in size or position. These positioning rules are defined as a fixed attachment to the Form widget, or to some other child widget, or to a relative position in the Form widget, or to the original position of the child widget.

Class xmFormWidgetClass
Superclasses BulletinBoard, Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fractionBase</td>
<td>integer</td>
<td>CSG</td>
<td>100</td>
</tr>
<tr>
<td>horizontalSpacing</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>rubberPositioning</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>verticalSpacing</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>bottomAttachment</td>
<td>atom</td>
<td>CSG</td>
<td>attach_none</td>
</tr>
<tr>
<td>bottomOffset</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>bottomPosition</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>bottomWidget</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>leftAttachment</td>
<td>atom</td>
<td>CSG</td>
<td>attach_none</td>
</tr>
<tr>
<td>leftOffset</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>leftPosition</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>leftWidget</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>resizable</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>rightAttachment</td>
<td>atom</td>
<td>CSG</td>
<td>attach_none</td>
</tr>
<tr>
<td>rightOffset</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>rightPosition</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>rightWidget</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>topAttachment</td>
<td>atom</td>
<td>CSG</td>
<td>attach_none</td>
</tr>
<tr>
<td>topOffset</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>topPosition</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>topWidget</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
</tbody>
</table>

fractionBase

Defines the relative position of a child widget within the Form widget. This applies only to children for which the value attachPosition has been specified for the sideAttachment resource (see the bottomAttachment resource, for example). The number specified with sidePosition will be divided by the value specified by fractionBase.

Value: integer
horizontalSpacing
Defines the distance between the right or left border of the child widget and the position of the element to which this border is attached.
Value: integer

rubberPositioning
Sets a default value for the positioning if no assignment has been made using sideAttachment. If the value of this resource is true, the position defined for the child widget by the X and Y coordinates will apply. If the value is false, the coordinate values 0,0 will apply.
Value: boolean values (true, false, on, off, yes, no)

verticalSpacing
Defines the distance between the top or bottom border of the child widget and the position of the element to which this border is attached.
Value: integer

Constraint resources for the attachment of widgets

bottomAttachment
Specifies the attachment for the bottom border of the child widget.
Value:

attach_none
The bottom border of the child widget will not be attached.

attach_form
The bottom border of the child widget will be attached to the bottom border of the Form widget.

attach_opposite_form
The bottom border of the child widget will be attached to the top border of the Form widget.

attach_widget
The bottom border of the child widget will be attached to the top border of the widget named by the bottomWidget resource.

attach_opposite_widget
The bottom border of the child widget will be attached to the bottom border of the widget named by the bottomWidget resource.

attach_position
The bottom border of the child widget will be attached to a position which is relative to the bottom border of the Form widget, and proportional to its height. This position is defined by the bottomPosition and fractionBase resources.
attach_self

The bottom border of the child widget is attached to a position which is proportional to the current Y coordinate of its bottom border divided by the height of the Form widget. This position is defined by the bottomPosition and fractionBase resources.

bottomOffset

Specifies the distance from the bottom border of the child widget to the element to which this border is attached.

Value: integer

bottomPosition

Defines the position of the bottom border of the child widget when the value set for the bottomAttachment resource is attach_position.

Value: integer

bottomWidget

Specifies the widget to which the bottom border of the child widget is to be attached.

Value: name of the widget

leftAttachment

Defines the attachment of the left-hand border of the child widget

Value: attach_none, attach_opposite_widget, attach_form, attach_opposite_form, attach_self, attach_widget

These values have the same meaning as the values for the bottomAttachment resource.

leftOffset

Specifies the distance from the left-hand border of the child widget to the element to which it is attached.

Value: integer

leftPosition

Defines the position of the left-hand border of the child widget when the value set for the bottomAttachment resource is attach_position.

Value: integer

leftWidget

Specifies the widget to which the left-hand border of the child widget is to be attached.

Value: name of the widget
Form Widgets

resizable
Specifies whether requests for a new size are permitted by the Form widget.
Value: boolean values (true, false, on, off, yes, no)

rightAttachment
Defines the attachment of the right-hand border of the child widget.
Value: attach_none, attach_opposite_widget, attach_form, attach_position, attach_opposite_form, attach_self, attach_widget
These values have the same meaning as the values for the bottomAttachment resource.

rightOffset
Specifies the distance from the right-hand border of the child widget to the element to which it is attached.
Value: integer

rightPosition
Defines the position of the right-hand border of the child widget when the value set for the bottomAttachment resource is attach_position.
Value: integer

rightWidget
Specifies the widget to which the right-hand border of the child widget is to be attached.
Value: name of the widget

topAttachment
Specifies the attachment for the top border of the child widget.
Value: attach_none, attach_opposite_widget, attach_form, attach_position, attach_opposite_form, attach_self, attach_widget
These values have the same meaning as the values for the bottomAttachment resource.

topOffset
Specifies the distance from the top border of the child widget to the element to which it is attached.
Value: integer

topPosition
Defines the position of the top border of the child widget when the value set for the bottomAttachment resource is attach_position.
Value: integer
<table>
<thead>
<tr>
<th>Widgets</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>topWidget</td>
<td>Specifies the widget to which the top border of the child widget is to be attached.</td>
</tr>
<tr>
<td>Value: name of the widget</td>
<td></td>
</tr>
</tbody>
</table>
Frame

The Frame widget is a simple Container widget which frames the window of exactly one child widget.

Class   xmFrameWidgetClass
Superclasses  Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>childType</td>
<td>atom</td>
<td>CSG</td>
<td>frame_workarea_child</td>
</tr>
<tr>
<td>childHorizontalAlignment</td>
<td>atom</td>
<td>CSG</td>
<td>alignment_beginning</td>
</tr>
<tr>
<td>childHorizontalSpacing</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>childVerticalAlignment</td>
<td>atom</td>
<td>CSG</td>
<td>alignment_center</td>
</tr>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>shadowType</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

marginHeight
The number of pixels between the top or bottom border and the child widget.
Value: integer

marginWidth
The number of pixels between the left or right border and the child widget.
Value: integer

shadowType
Defines the type of shadow for the widget.
Value: shadow_in  The widget appears to be sunk into the surface.
shadow_out       The widget appears raised relative to the surface.

The following two values use double shadows to create a three-dimensional impression.
shadow_etched_in  The widget appears to be sunk into the surface.
shadow_etched_out The widget appears raised relative to the surface.

Constraint resources for child widgets

childType
Specifies whether a child is a title or work area.
Value: frame_title_child, frame_workarea_child, or frame_generic_child
childHorizontalAlignment
  Specifies the alignment of the title.
  Value: alignment_beginning, alignment_center, or alignment_end

childHorizontalSpacing
  Specifies the minimum distance between edge of title text and Frame shadow.

childVerticalAlignment
  Specifies the vertical alignment of the title text, or the title area in relation to the top shadow of the Frame.
  Value: alignment_baseline_bottom  alignment_baseline_top
         alignment_widget_top       alignment_center
         alignment_widget_bottom
**Gadget**

The xmGadgetClass class is used as a superclass for all classes of gadget.

Class: xmGadgetClass
Superclasses: RectObj, Object

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottomShadowColor</td>
<td>atom</td>
<td>G</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>helpCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>highlightColor</td>
<td>atom</td>
<td>G</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>highlightOnEnter</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>highlightThickness</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>navigationType</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>shadowThickness</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>topShadowColor</td>
<td>atom</td>
<td>G</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>traversalOn</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>unitType</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>userData</td>
<td>atom</td>
<td>CSG</td>
<td>''</td>
</tr>
</tbody>
</table>

- **bottomShadowColor**
  Specifies the color used to draw the bottom and right side of the border shadow.
  
  Value: atom

- **helpCallback**
  Specifies the callback list which will be called if the help key sequence is pressed.
  
  Value: can only be set by xtAddCallback/4

- **highlightColor**
  Specifies the color used to draw the highlighting rectangle.

  Value: atom

- **highlightOnEnter**
  Determines whether the border is drawn highlighted or not.

  Value: boolean values (true, false, on, off, yes, no)

- **highlightThickness**
  Defines the width, in pixels, of the highlighted border.

  Value: integer
Widgets Gadget

navigationType
Specifies the navigation group to which the gadget belongs. The navigation group defines the scope of user control permitted via the keyboard.
Value:

none
The widget does not belong to any navigation group.
tab_group
The widget is automatically incorporated into keyboard operations unless AddTabGroup has been called.
sticky_tab_group
The widget is automatically incorporated into keyboard operations even if xmAddTabGroup has been called.
exclusive_tab_group
The application expressly incorporates the widget into keyboard operations.

shadowThickness
Specifies the width, in pixels, of the shadowed border.
Value: integer

topShadowColor
Specifies the color used to draw the top and left sides of the border shadow.
Value: atom

traversalOn
Switches traversal on and off. The value true for traversal means that control via the keyboard is permitted. The key bindings (translations) are dependent on the widget class which is being used. The way in which key events are passed on depends on the entries in the tab groups.
Value: boolean values (true, false, on, off, yes, no)

unitType
Defines a unit for enlargements, reductions and positioning.
Value: pixels, 100th_millimeters, 1000th_inches, 100th_points, 100th_font_units

userData
Binds specific user data to the gadget.
Value: any atom
Gadget Widgets

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>helpCallback</td>
<td>help</td>
<td>Help key has been pressed</td>
</tr>
</tbody>
</table>

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.
Label (widget)

A Label widget is used to output title lines or text items. A Label widget contains either text or a pixmap. Push buttons or toggle buttons are subclasses of the Label widget.

Class xmLabelWidgetClass
Superclasses Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>accelerator</td>
<td>atom</td>
<td>CSG</td>
<td>’ ’</td>
</tr>
<tr>
<td>acceleratorText</td>
<td>atom, list</td>
<td>CSG</td>
<td>’ ’</td>
</tr>
<tr>
<td>alignment</td>
<td>atom</td>
<td>CSG</td>
<td>alignment_center</td>
</tr>
<tr>
<td>fontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>labelInsensitivePixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>labelPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>labelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>labelType</td>
<td>atom</td>
<td>CSG</td>
<td>string</td>
</tr>
<tr>
<td>marginBottom</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>marginLeft</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>marginRight</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>marginTop</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>mnemonic</td>
<td>atom</td>
<td>CSG</td>
<td>’ ’</td>
</tr>
<tr>
<td>mnemonicCharSet</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>recomputeSize</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>stringDirection</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

accelerator
Specifies an accelerator table if the button is used for a menu item.
Value: valid accelerator table

acceleratorText
Specifies the text which will be output for the accelerator. This text will be displayed on the side of the label indicated by the alignment resource. This is only used with push buttons, toggle buttons and menus.
Value: valid XmString

alignment
Defines how the text or pixmap is justified.
Value: alignment_center centered
alignment_end right-justified
alignment_beginning left-justified
fontList
  Specifies the font for text items.
  Value: valid font list

labelInsensitivePixmap
  Specifies a pixmap to be used for displaying the button when the labelType resource is set to the value pixmap and the button cannot currently be activated, i.e. it is shown grayed out.
  Value: pixmap (name of a bitmap file)

labelPixmap
  Specifies a pixmap to be used for displaying the button when the labelType resource is set to the value pixmap.
  Value: pixmap (name of a bitmap file)

labelString
  Defines a character string used as the label on the button when the labelType resource is set to the value string.
  Value: valid XmString

labelType
  Specifies the type of representation to be used for the button.
  Value: pixmap (string)

marginBottom
  Defines the distance, in pixels, from the bottom border of the label to the bottom boundary.
  Value: integer

marginHeight
  Defines the distance, in pixels, from the top border of the label to the top shadow.
  Value: integer

marginLeft
  Defines the distance, in pixels, from the label to the left-hand boundary.
  Value: integer

marginRight
  Defines the distance, in pixels, from the label to the right-hand boundary.
  Value: integer

marginTop
  Defines the distance, in pixels, from the label to the top boundary.
  Value: integer
Widgets

marginWidth

Defines the distance, in pixels, from the label to the left- or right-hand shadow.
Value: integer

mnemonic

Specifies a character (key symbol) to be used as the mnemonic. If this character occurs in a component part of the XmString labelString, it will be underlined on its first occurrence if the component part belongs to the charset mnemonicCharSet.
Value: single-character atom

mnemonicCharSet

Specifies which component part of the XmString labelString is to be searched for the occurrence of mnemonic.
Value: valid character set

recomputeSize

Specifies whether the widget will be matched to the size of the character string or pixmap.
Value: boolean values (true, false, on, off, yes, no)

stringDirection

Specifies the direction in which the character string is to be output.
Value:  
  - string_direction_l_to_r  from left to right
  - string_direction_r_to_l  from right to left
Label (gadget)

The Label gadget corresponds to the Label widget. When a Label gadget is created, the `xmLabelGadgetClass` class must be used.

Class: `xmLabelGadgetClass`
Superclasses: `Gadget`, `RectObj`, `Object`

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.
List

The List widget can be used to output lists from which the user can make a selection.

Class xmListWidgetClass
Superclasses Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>automaticSelection</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>browseSelectionCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>defaultActionCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>doubleClickInterval</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>extendedSelectionCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>fontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>horizontalScrollbar</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>itemCount</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>items</td>
<td>list</td>
<td>CSG</td>
<td>[ ]</td>
</tr>
<tr>
<td>listMarginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>listMarginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>listSizePolicy</td>
<td>atom</td>
<td>CG</td>
<td>variable</td>
</tr>
<tr>
<td>listSpacing</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>multipleSelectionCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>scrollBarDisplayPolicy</td>
<td>atom</td>
<td>CSG</td>
<td>as_needed</td>
</tr>
<tr>
<td>selectedItemCount</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>selectedItems</td>
<td>list</td>
<td>CSG</td>
<td>[ ]</td>
</tr>
<tr>
<td>selectionPolicy</td>
<td>atom</td>
<td>CSG</td>
<td>browse_select</td>
</tr>
<tr>
<td>singleSelectionCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>stringDirection</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>topItemPosition</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>visibleItemCount</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>verticalScrollBar</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
</tbody>
</table>

automaticSelection

Defines whether an entry in the list will be marked if the pointer is moved onto it.

Value: boolean values (true, false, on, off, yes, no)

browseSelectionCallback

Specifies the callback list to be called if several entries are marked in succession. This will apply if the selectionPolicy resource has the value browse_select.

Value: can only be set by xtAddCallback/4
defaultActionCallback
   Specifies the callback list to be called if the user clicks on an entry twice.
   Value: can only be set by xtAddCallback/4

doubleClickInterval
   Specifies the maximum time interval (in milliseconds) that may elapse between two successive clicks of a mouse button for this to be accepted as a double click.
   Value: integer

extendedSelectionCallback
   Specifies the callback list to be called if several entries are marked. This will apply if the selectionPolicy resource has the value extended select.
   Value: can only be set by xtAddCallback/4

fontList
   Specifies the font to be used when outputting the entries in the list.
   Value: valid font list

horizontalScrollbar
   Specifies which widget is to be used as the horizontal scroll bar.
   Value: name of the widget

itemCount
   The number of entries in the list. The value is set automatically to the length of the items resource list.
   Value: integer

items
   A list of the entries.
   Value: list of XmStrings

listMarginHeight
   Specifies the height, in pixels, of the spacing between the top and bottom borders of the list and the entries.
   Value: integer

listMarginWidth
   Specifies the width, in pixels, of the spacing between the right and left borders of the list and the entries.
   Value: integer

listSizePolicy
   Specifies the list behavior if an entry is longer than the current width of the list permits.
   Value:
constant
   The size of the list remains constant; a horizontal scroll bar is appended.
variable
   The size of the list adjusts to the size of the longest entry.
resize_if_possible
   If possible, the list adjusts to the size of the longest entry. If the entry
   extends beyond the maximum size of the widget, a horizontal scroll bar
   is added.

listSpacing
   Specifies the spacing between the entries (in pixels).
Value: integer

multipleSelectionCallback
   Specifies the callback list to be called if several entries are marked. This will
   apply if the selectionPolicy resource has the value multiple_select.
Value: can only be set by xtAddCallback/4

scrollBarDisplayPolicy
   Specifies the display policy for a vertical scroll bar.
Value: as_needed A vertical scroll bar will be displayed
      only when required.
      static A vertical scroll bar is always dis-
      played.

selectedItemCount
   Specifies the number of entries marked in the list of marked entries. The
   value is set automatically to the length of the selectedItems resource list.
Value: integer

selectedItems
   Contains the list of marked entries.
Value: list of XmStrings

selectionPolicy
   Defines the way in which the user can mark the entries.
Value:
   single_select
      Only one entry can be marked. All other markings will be canceled.
   multiple_select
      More than one entry may be marked. Each is marked individually by
      clicking on it.
   browse_select
      More than one entry may be marked, either by clicking on them indi-
      videntally or also by dragging the pointer across a number of entries.
extended_select
More than one entry may be marked by dragging the pointer across a number of entries with the button held down. If the user then clicks on another entry, the markings are canceled. However, if the CTRL key is pressed at the same time, the previous markings will be retained.

singleSelectionCallback
Specifies the callback list to be called if a single entry is marked. This will apply if the resource selectionPolicy has the value single_select.
Value: can only be set by xtAddCallback/4

stringDirection
Specifies the direction in which character strings are to be output.
Value: string_direction_l_to_r from left to right
string_direction_r_to_l from right to left

topItemPosition
Defines the position of the first visible entry in the list.
Value: integer

verticalScrollBar
Specifies which widget is to be used as the vertical scroll bar.
Value: name of the widget

visibleItemCount
Specifies the number of visible entries.
Value: integer
### Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>browseSelectionCallback</td>
<td>browse_select</td>
<td>Browse type marking</td>
</tr>
<tr>
<td>defaultActionCallback</td>
<td>default_action</td>
<td>Marking by double click</td>
</tr>
<tr>
<td>extendedSelectionCallback</td>
<td>extended_select</td>
<td>Extended type marking</td>
</tr>
<tr>
<td>multipleSelectionCallback</td>
<td>multiple_select</td>
<td>Multiple type marking</td>
</tr>
<tr>
<td>singleSelectionCallback</td>
<td>single_select</td>
<td>Single type marking</td>
</tr>
</tbody>
</table>

### Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>list</td>
<td>all</td>
<td>Last marked entry</td>
</tr>
<tr>
<td>item_position</td>
<td>integer</td>
<td>all</td>
<td>Position of the entry in the list</td>
</tr>
<tr>
<td>selected_items</td>
<td>list</td>
<td>browse_select extended_select multiple_select</td>
<td>List of the marked entries</td>
</tr>
<tr>
<td>selected_item_count</td>
<td>list</td>
<td>browse_select extended_select multiple_select</td>
<td>Number of marked entries</td>
</tr>
<tr>
<td>selected_items_position</td>
<td>list</td>
<td>browse_select extended_select multiple_select</td>
<td>Positions of the marked entries</td>
</tr>
<tr>
<td>selection_type</td>
<td>atom</td>
<td>extended_select</td>
<td>Type of marking</td>
</tr>
</tbody>
</table>

Motif Manual 98 IF/Prolog V5.2
MainWindow

The MainWindow widget provides a standard layout for the first window of an application.

Class       xmMainWindowWidgetClass
Superclasses ScrolledWindow, Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandWindow</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>commandWindowLocation</td>
<td>atom</td>
<td>CG</td>
<td>command_above_workspace</td>
</tr>
<tr>
<td>mainWindowMarginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>mainWindowMarginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>menuBar</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>messageWindow</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>showSeparator</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
</tbody>
</table>

commandWindow

Defines the widget to be used as the CommandWindow. This must be a child of the MainWindow widget.

Value: name of the widget

commandWindowLocation

Specifies the position of the CommandWindow widget.

Value:

command_above_workspace
Positions the CommandWindow widget between the menu bar and the work window.

command_above_workspace
Positions the CommandWindow widget between work window and the message window.

mainWindowMarginHeight

Specifies the height, in pixels, of the margin for the upper and lower border.

Value: integer

mainWindowMarginWidth

Specifies the width, in pixels, of the margin for the right and left border.

Value: integer

menuBar

Specifies which widget is to be used as the menu bar.

Value: name of the widget
Widgets

MainWindow

messageWindow
  Specifies which widget is to be used as the message window.
  Value: name of the widget

showSeparator
  Draws separators between the components of the MainWindow widget.
  Value: boolean values (true, false, on, off, yes, no)
Manager

The Manager class of widgets is a superclass for other widget classes. It provides resources and methods for use in determining the appearance of child widgets.

Class         xmManagerWidgetClass
Superclasses  Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottomShadowColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>bottomShadowPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>foreground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>helpCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>highlightColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>highlightPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>initialFocus</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>navigationType</td>
<td>atom</td>
<td>CSG</td>
<td>tab_group</td>
</tr>
<tr>
<td>shadowThickness</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>stringDirection</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>topShadowColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>topShadowPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>traversalOn</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>unitType</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>userData</td>
<td>atom</td>
<td>CSG</td>
<td>' '</td>
</tr>
</tbody>
</table>

bottomShadowColor

Specifies the color of the shadows on the bottom and right-hand borders.
Value: name of a valid color

bottomShadowPixmap

Specifies the pattern of the shadows on the bottom and right-hand borders.
Value: pixmap (name of a bitmap file)

foreground

Specifies the color of the foreground for outputs.
Value: name of a valid color

helpCallback

Specifies the callback list to be called when the help key sequence is pressed.
Value: can only be set by xtAddCallback/4

highlightColor

Specifies the color to use for highlighting.
Value: name of a valid color
highlightPixmap
  Specifies the pattern to be used for highlighting.
  Value: pixmap (name of a bitmap file)

initialFocus
  Specifies the name of a widget descendant of the manager.
  Value: atom, name of a widget

navigationType
  Specifies the navigation group to which the widget belongs.
  Value:

  none
    The widget does not belong to any navigation group.

  tab_group
    The widget is automatically incorporated into keyboard operations unless xmAddTabGroup has been called.

  sticky_tab_group
    The widget is automatically incorporated into keyboard operations even if xmAddTabGroup has been called.

  exclusive_tab_group
    The application expressly incorporates the widget into keyboard operations.

shadowThickness
  Specifies the width of the shadow border.
  Value: integer

stringDirection
  Specifies the direction in which character strings are to be output.
  Value: string_direction_l_to_r from left to right
         string_direction_r_to_l from right to left

topShadowColor
  Specifies the color for the shadows on the top and left borders.
  Value: name of a valid color

topShadowPixmap
  Specifies the pattern of the shadows on the top and left-hand borders.
  Value: pixmap (name of a bitmap file)

traversalOn
  Switches traversal on and off. The value true for traversal means that control via the keyboard is permissible. The key bindings (translations) are dependent on the widget class which is being used. The way in which key events are passed on depends on the entries in the tab groups.
Manager: Widgets

Value: boolean values (true, false, on, off, yes, no)

**unitType**

Defines a unit for enlargements, reductions and positioning.

Value: pixels Pixels

100th millimeters Hundredths of a millimeter

1000th inches Thousandths of an inch

100th points Hundredths of a point (1 point = 1/72 inch)

100th font units Hundredths of a font unit

**userData**

Binds specific user data to the widget.

Value: any atom

**Callbacks**

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>helpCallback</td>
<td>help</td>
<td>Help key pressed</td>
</tr>
</tbody>
</table>
MenuShell

The MenuShell widget is a Container widget for pop-up or pull-down menus.

Class xmMenuShellWidgetClass
Superclasses OverrideShell, Shell, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttonFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>defaultFontList</td>
<td>atom, list</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>labelFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

buttonFontList
Specifies the font for button descendants.
Value: valid font list

defaultFontList
Specifies the font for child widgets.
Value: valid font list

labelFontList
Specifies the font for label descendants.
Value: valid font list
MessageBox

The MessageBox widget is a widget for message dialogs.

Class    xmMessageBoxWidgetClass
Superclasses BulletinBoard, Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>cancelLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Cancel'</td>
</tr>
<tr>
<td>defaultButtonType</td>
<td>atom</td>
<td>CSG</td>
<td>dialog_ok_button</td>
</tr>
<tr>
<td>dialogType</td>
<td>atom</td>
<td>CSG</td>
<td>message</td>
</tr>
<tr>
<td>helpLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Help'</td>
</tr>
<tr>
<td>messageAlignment</td>
<td>atom</td>
<td>CSG</td>
<td>alignment_beginning</td>
</tr>
<tr>
<td>messageString</td>
<td>atom, list</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>minimizeButtons</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>okCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>okLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'OK'</td>
</tr>
<tr>
<td>symbolPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(depends on dialogType)</td>
</tr>
</tbody>
</table>

cancelCallback
   Specifies the callback list to be called when the user activates the cancel button.
   Value: can only be set by xtAddCallback/4

cancelLabelString
   Defines a character string to be used for labeling the cancel button.
   Value: valid XmString

defaultButtonType
   Specifies which type of button is to be used.
   Value: dialog_cancel_button   dialog_ok_button
           dialog_help_button   dialog_none

dialogType
   Specifies which type of dialog box is to be used.
   Value: dialog_error      Dialog box for error messages
           dialog_information| Dialog box for information items
           dialog_message    | Dialog box for messages
           dialog_question   | Dialog box for questions
           dialog_warning    | Dialog box for warnings
           dialog_working    | Dialog box for messages about the progress of work
helpLabelString
  Defines the character string to be output as the label on the Help button.
  Value: valid XmString

messageAlignment
  Specifies how the message texts are to be aligned.
  Value: alignment_beginning left-justified
          alignment_center centered
          alignment_end right-justified

messageString
  Defines the character string to be output as the message text.
  Value: valid XmString

minimizeButtons
  Specifies whether each button can determine its own size (true), or whether
  all the buttons are to be matched to the size of the largest (false).
  Value: boolean values (true, false, on, off, yes, no)

okCallback
  Specifies the callback list to be called if the user activates the OK button.
  Value: can only be set by xtAddCallback/4

okLabelString
  Defines the character string to be used as the label on the OK button.
  Value: valid XmString

symbolPixmap
  Specifies which pattern is to be used as the symbol representing messages.
  Value: pixmap (name of a bitmap file)

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelCallback</td>
<td>cancel</td>
<td>Cancel button activated</td>
</tr>
<tr>
<td>okCallback</td>
<td>ok</td>
<td>OK button activated</td>
</tr>
<tr>
<td>unmapCallback</td>
<td>unmap</td>
<td>Parent dialog shell deactivated</td>
</tr>
</tbody>
</table>

See also

SelectionBox
OverrideShell

OverrideShell widgets are not managed by the Window Manager. They can be used, for example, to place a menu at a particular position on the screen.

Class overrideShellWidgetClass
Superclasses Shell, Composite, Core

overrideShellWidgetClass does not define any resources of its own.

See also

WMShell
PanedWindow

The PanedWindow widget is a Container widget which arranges its child widgets vertically above one another. All the child widgets have the same width. The boundary between two child widgets can be moved. The boundary is indicated by a rectangle (sash) which is used in moving the boundary. If required, a separating line will be displayed between the two windows. When the line separating two widgets is moved, the effect is to increase the height of one of them and to reduce the height of the other.

Class        xmPanedWindowWidgetClass
Superclasses Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>3</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>3</td>
</tr>
<tr>
<td>refigureMode</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>sashHeight</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>sashIndend</td>
<td>integer</td>
<td>CSG</td>
<td>-10</td>
</tr>
<tr>
<td>sashShadowThickness</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>sashWidth</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>separatorOn</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>spacing</td>
<td>integer</td>
<td>CSG</td>
<td>8</td>
</tr>
</tbody>
</table>

Constraint resources for child widgets

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowResize</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>paneMaximum</td>
<td>integer</td>
<td>CSG</td>
<td>1000</td>
</tr>
<tr>
<td>paneMinimum</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>positionIndex</td>
<td>atom</td>
<td>CSG</td>
<td>last_position</td>
</tr>
<tr>
<td>skipAdjust</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
</tbody>
</table>

marginHeight

Defines the distance, in pixels, between the top or bottom border of the widget and the child widgets.

Value: integer

marginWidth

Defines the distance, in pixels, between the right and left border of the widget and the child widgets.

Value: integer
PanedWindow Widgets

refigureMode  Specifies whether the positions of the window panes are to be recalculated and the panes repositioned, if the user has made changes to the PanedWindow widget. If this resource has the value true, the positions of the children will be adjusted to match the position of the widget.
Value: boolean values (true, false, on, off, yes, no)

sashHeight  Specifies the height, in pixels, of the rectangular sash which can be used to move the boundary between two child widgets.
Value: integer

sashIndent  Specifies the horizontal position of the rectangular sash. Positive values relate to the distance from the left-hand border, negative ones to the distance from the right border.
Value: integer

sashShadowThickness  Specifies the width of the shadow for the rectangular sash.
Value: integer

sashWidth  Specifies the width, in pixels, of the rectangular sash.
Value: integer

separatorOn  Specifies whether separators are output between the components of the widget.
Value: boolean values (true, false, on, off, yes, no)

spacing  Specifies the distance in pixels between the panes for the child widgets.
Value: integer

Constraint resources for child widgets

allowResize  Specifies whether the sizes of the child widgets can be changed by a program.
Value: boolean values (true, false, on, off, yes, no)

paneMaximum  Defines the maximum height of the child widget, in pixels.
Value: integer
Widgets

paneMinimum

Defines the minimum height of the child widget, in pixels.
Value: integer

positionIndex

Specifies the position of the widget in its parent’s list of children.
Value: integer or last_position

skipAdjust

Specifies whether the size of the child widget may be adjusted automatically by the PanedWindow widget. The value true defines that the size will not be adjusted automatically.
Value: boolean values (true, false, on, off, yes, no)

See also

RowColumn, ScrolledWindow, BulletinBoard, Frame
Primitive

The Primitive widget is a superclass for other widget classes. Among other functions, it handles the drawing of frames, the creation of highlights and other elements which produce the three-dimensional effect. This class of widgets cannot be instantiated.

Class: xmPrimitiveWidgetClass
Superclasses: Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottomShadowColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>bottomShadowPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>foreground</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>helpCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>highlightColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>highlightOnEnter</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>highlightPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>highlightThickness</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>navigationType</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>shadowThickness</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>topShadowColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>topShadowPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>traversalOn</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>unitType</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>userData</td>
<td>atom</td>
<td>CSG</td>
<td>,</td>
</tr>
</tbody>
</table>

**bottomShadowColor**

Specifies the color of the shadows on the bottom and right-hand borders.

Value: name of a valid color

**bottomShadowPixmap**

Specifies the pattern for the shadows on the bottom and right-hand borders.

Value: pixmap (name of a bitmap file)

**foreground**

Specifies the color of the foreground for outputs.

Value: name of a valid color

**helpCallback**

Specifies the callback list to be called if the help key sequence is pressed.

Value: can only be set by xtAddCallback/4
Widgets

**highlightColor**

Specifies the color to be used for highlighting.

Value: name of a valid color

**highlightOnEnter**

Specifies whether the input window is highlighted.

Value: boolean values (true, false, on, off, yes, no)

**highlightPixmap**

Specifies the pattern to use for highlighting.

Value: pixmap (name of a bitmap file)

**shadowThickness**

Specifies the width of the shadow in pixels.

Value: integer

**topShadowColor**

Specifies the color for the shadows on the top and left borders.

Value: name of a valid color

**topShadowPixmap**

Specifies the pattern for the shadows on the top and left-hand borders.

Value: pixmap (name of a bitmap file)

**traversalOn**

Switches traversal on and off. The value true for traversal means that control via the keyboard is permissible. The key bindings (translations) are dependent on the widget class which is being used. The way in which key events are passed on depends on the entries in the tab groups.

Value: boolean values (true, false, on, off, yes, no)

**unitType**

Defines a unit for enlargements, reductions and positioning.

Value: pixels, 100th_millimeters, 1000th_inches, 100th_points, 100th_font_units

<table>
<thead>
<tr>
<th>Unit Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>Pixels</td>
</tr>
<tr>
<td>100th_millimeters</td>
<td>Hundredths of a millimeter</td>
</tr>
<tr>
<td>1000th_inches</td>
<td>Thousandths of an inch</td>
</tr>
<tr>
<td>100th_points</td>
<td>Hundredths of a point (1 point = 1/72 inch)</td>
</tr>
<tr>
<td>100th_font_units</td>
<td>Hundredths of a font unit</td>
</tr>
</tbody>
</table>

**userData**

Binds specific user data to the widget.

Value: any atom
## Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>helpCallback</td>
<td>help</td>
<td>Help key pressed</td>
</tr>
</tbody>
</table>
Widgets

PushButton

PushButton (widget)

The PushButton widget is a button which is used for executing actions. A push button consists of a text item or a pixmap, surrounded by a shadowed border.

Class xmPushButtonWidgetClass
Superclasses Label, Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>armCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>armColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>armPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultButtonShadowThickness</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>fillOnArm</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>multiClick</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>showAsDefault</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
</tbody>
</table>

activateCallback
Specifies the callback list which will be called if the button is activated.
Value: can only be set by xtAddCallback/4

armCallback
Specifies the callback list which will be called if the button is marked.
Value: can only be set by xtAddCallback/4

armColor
Specifies the color to be used for filling the button when it is marked.
Value: name of a valid color

armPixmap
Specifies the pattern to be used for displaying the button when it is marked.
Value: pixmap (name of a bitmap file)

defaultButtonShadowThickness
Defines the width of the shadow for the button. If this value is set to 0, the value of the showAsDefault resource will be used.
Value: integer
_pushButton Widgets

**PushButton**

### disarmCallback

Specifies the callback list which will be called if the marking of the button is canceled.

Value: can only be set by xtAddCallback/4

### fillOnArm

Specifies whether the button is to be filled with the color specified by arm-Color when the button is marked. If the button is part of a menu, this resource will be ignored.

Value: boolean values (true, false, on, off, yes, no)

### multiClick

Specifies whether multiple clicking on the button in rapid succession will result in multiple execution of the function or not. If the button is not within a menu, the default value is multiclick_keep.

Value:  
- multiclick_discard: only the first click is processed
- multiclick_keep: all clicks will be processed

### showAsDefault

Specifies the width of a second type of surround, which will be used to mark the button if it is activated by the user pressing the Return key.

Value: integer

### Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>activate</td>
<td>Input focus accepted</td>
</tr>
<tr>
<td>armCallback</td>
<td>arm</td>
<td>Parent dialog shell marked</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>disarm</td>
<td>Parent dialog shell marking canceled</td>
</tr>
</tbody>
</table>

### Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>click_count</td>
<td>integer</td>
<td>activate</td>
<td>Number of clicks</td>
</tr>
</tbody>
</table>
PushButton (gadget)

The PushButton gadget corresponds to the PushButton widget. The class xmPushButton-GadgetClass must be used when creating a PushButton gadget.

Class                  xmPushButtonGadgetClass
Superclasses           Gadget, RectObj, Object

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.

See also

DrawnButton, ToggleButton, CascadeButton
RowColumn

The RowColumn widget is a Container widget which is used to organize the management of position and size of child widgets in rows and columns.

**Class** \( \text{xmRowColumnWidgetClass} \)

**Superclasses** Manager, Constraint, Composite, Core

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustLast</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>adjustMargin</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>entryAlignment</td>
<td>atom</td>
<td>CSG</td>
<td>alignment_beginning</td>
</tr>
<tr>
<td>entryBorder</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>entryCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>entryClass</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>entryVerticalAlignment</td>
<td>atom</td>
<td>CSG</td>
<td>alignment_center</td>
</tr>
<tr>
<td>isAligned</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>isHomogeneous</td>
<td>atom</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>labelString</td>
<td>atom, list</td>
<td>C</td>
<td>' '</td>
</tr>
<tr>
<td>mapCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>menuAccelerator</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>menuHelpWidget</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>menuHistory</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>menuPost</td>
<td>atom</td>
<td>CSG</td>
<td>' '</td>
</tr>
<tr>
<td>mnemonic</td>
<td>atom</td>
<td>CSG</td>
<td>' '</td>
</tr>
<tr>
<td>mnemonicCharSet</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>numColumns</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>orientation</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>packing</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>popupEnabled</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>radioAlwaysOne</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>radioBehavior</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>resizeHeight</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>resizeWidth</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>rowColumnType</td>
<td>atom</td>
<td>CG</td>
<td>work_area</td>
</tr>
<tr>
<td>spacing</td>
<td>integer</td>
<td>CGS</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>subMenuId</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
</tbody>
</table>
Widgets

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tearOffMenuActivateCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>tearOffMenuDeactivateCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>tearOffModel</td>
<td>atom</td>
<td>CSG</td>
<td>tear_off_disabled</td>
</tr>
<tr>
<td>unmapCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>whichButton</td>
<td>atom</td>
<td>CGS</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

Resources for the creation of simple menus with convenience predicates

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttonAccelerators</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>buttonAcceleratorText</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>buttonCount</td>
<td>integer</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>buttonMnemonicCharSets</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>buttonMnemonics</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>buttons</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>buttonSet</td>
<td>integer</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td>buttonType</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>optionLabel</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>optionMnemonic</td>
<td>atom</td>
<td>C</td>
<td>''</td>
</tr>
<tr>
<td>postFromButton</td>
<td>integer</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td>simpleCallback</td>
<td>atom,list</td>
<td>C</td>
<td>''</td>
</tr>
</tbody>
</table>

adjustLast

Specifies whether or not the last line of child widgets is extended down to the bottom border (horizontal orientation), or the last column of child widgets is extended to the right-hand border (vertical orientation).

Value: boolean values (true, false, on, off, yes, no)

adjustMargin

Specifies whether the bounds for all entries in the RowColumn widget will be adjusted to the same values.

Value: boolean values (true, false, on, off, yes, no)

entryAlignment

Specifies the alignment of the labels. This applies to child widgets of the Label type, or their subclasses.

Value: alignment

- alignment_beginning: left-justified
- alignment_center: centered
- alignment_end: right-justified

entryBorder

Specifies a uniform border width for all child widgets, in pixels. If this value is set to 0, the border width is not uniform.

Value: integer
**entryCallback**
Maps the activateCallback and the valueChangedCallback for all child widgets to the entryCallback.
Value: can only be set by xtAddCallback/4

**entryClass**
Specifies the widget class for child widgets when they must all belong to the same class. This is the case when the isHomogeneous resource has the value true.
Value: name of the widget class

**entryVerticalAlignment**
Specifies the type of vertical alignment of label and text widgets.
Value: alignment_baseline_bottom  alignment_baseline_top
alignment_contents_bottom  alignment_center
alignment_contents_top

**isAligned**
Specifies whether entries are to have uniform alignment. This applies to child widgets of the Label type or subclasses thereof.
Value: boolean values (true, false, on, off, yes, no)

**isHomogeneous**
Specifies whether all the child widgets must belong to the same widget class. The class is specified by the entryClass resource.
Value: boolean values (true, false, on, off, yes, no)

**labelString**
Defines a character string for menu items. This is only meaningful if the value of the rowColumnType resource is set to menu_option.
Value: valid XmString

**mapCallback**
Specifies the callback list to be called if the window which contains the RowColumn widget is to be mapped on the screen. This applies only if the parent widget is a DialogShell widget.
Value: can only be set by xtAddCallback/4

**marginHeight**
Defines the distance, in pixels, between the top and bottom border and the first entry in each column.
Value: integer

**marginWidth**
Defines the distance, in pixels, between the right- and left-hand border and the first entry in each line.
menuAccelerator

Specifies a key sequence by which a menu can be accessed. This resource is only meaningful when the RowColumn widget is used for a pop-up menu or a menu bar. The values for this resource must be specified in the same format as the key symbols in the accelerator and translation tables. In the case of a pop-up menu, if the user enters the key sequence specified as the accelerator, the pop-up menu will be posted. In the case of a menu bar, if the user enters the key sequence specified as the accelerator, the first entry in the menu bar will be highlighted and the traversalOn resource will be set to true. For a pop-up menu, the preset value of this resource is menu, and for a menu bar it is menuBar. The accelerator is switched off by setting the popupEnabled resource to false.

Value: atom

menuHelpWidget

Specifies which widget is to be used as the Help widget for menu_bar.

Value: name of the widget

menuHistory

The last menu item which was activated.

Value: name of the widget

menuPost

Specifies the event which leads to the posting of a menu.

Value: atom

mnemonic

Specifies a character (key symbol) to be used as the mnemonic. If this character occurs in a component part of the XmString labelString, it will be underlined on its first occurrence if the component part belongs to the charset mnemonicCharSet. This is only meaningful if the value of the rowColumnType resource is set to menu_option.

Value: single-character atom

mnemonicCharSet

Specifies which component part of the XmString labelString is to be searched for the occurrence of mnemonic.

Value: valid character set

numColumns

Specifies the number of extensions to the entries. This resource is only meaningful if the value of the packing resource is set to pack_column.

Value: integer
RowColumn Widgets

orientation

Specifies whether the table is to be arranged in rows (horizontally) or columns (vertically).

Value: horizontal for a menu bar
vertical in other cases

packing

Specifies how the child widgets in a RowColumn widget can be grouped together.

Value:

pack_tight
Arranges the child widgets closely beside each other in a line. If there is no space left, a new line will be started. The heights or widths of the child widgets will be made equal.

pack_column
The boxes for all the child widgets are the same size. The size of the boxes will be determined by the largest child widget.

pack_none
Each child widget will be positioned as specified for it by the x and y resources.

popupEnabled

Specifies whether inputs are permitted using the keyboard, accelerators or mnemonics.

Value: boolean values (true, false, on, off, yes, no)

radioAlwaysOne

Specifies whether the last active toggle button is to remain active as long as no other button is activated.

Value: boolean values (true, false, on, off, yes, no)

radioBehavior

Specifies whether or not all child widgets which are toggle buttons will behave as radio buttons (i.e. that there can only be one button active at any time). If the value is true, entryClass must be set, so that the X Window System does not generate any warnings.

Value: boolean values (true, false, on, off, yes, no)

resizeHeight

Specifies whether a new height will be requested when necessary.

Value: boolean values (true, false, on, off, yes, no)

resizeWidth

Specifies whether a new width will be requested when necessary.

Value: boolean values (true, false, on, off, yes, no)
Widgets

RowColumn

**rowColumnType**

Specifies which type of RowColumn will be created.

**Value:**
- `work_area` A work area
- `menu_bar` A menu bar
- `menu_pulldown` A pull-down menu
- `menu_popup` A pop-up menu
- `menu_option` A menu item

**spacing**

Specifies the horizontal and vertical distance between items, in pixels.

**Value:** integer

**subMenuId**

Specifies the MenuPane widget for a menu item. This is only meaningful if the value set for the `rowColumnType` resource is `menu_option`.

**Value:** name of the widget

**tearOffMenuActivateCallback**

Specifies the callback list which will be called when the tear-off MenuPane is about to be activated.

**Value:** can only be set by xtAddCallback/4

**tearOffMenuActivateCallback**

Specifies the callback list which will be called when the tear-off MenuPane is about to be deactivated.

**Value:** can only be set by xtAddCallback/4

**unmapCallback**

Defines the callback list which is to be called if the mapping of the widget is canceled. This is only applicable if the parent widget is a DialogShell widget.

**Value:** can only be set by xtAddCallback/4

**whichButton**

Specifies which mouse button is associated with a menu.

**Value:** `button1`, `button2`, `button3`, `button4`, `button5`

**Resources for the creation of simple menus with convenience predicates**

**buttonAccelerators**

Specifies a list of accelerators for the buttons created. This list contains one element for each button, separator and title created.

**Value:** atom, list
buttonAcceleratorText
  Specifies a list of compound strings which are output for the accelerators
  of the buttons created. This list contains one element for each button,
  separator and title created.
  Value: atom, list

buttonCount
  Specifies the number of menu buttons, separators and titles to be created.
  The value is set automatically to the length of the buttons resource list.
  Value: integer

buttonMnemonicCharSets
  Specifies a list of symbolic character set names for the output of mnemonics
  for buttons. This list contains one element for each button, separator and
  title created. The default setting is dependent on the widget’s environment.
  Value: atom, list

buttonMnemonics
  Specifies a list of mnemonics for the buttons created. This list contains one
  element for each button, separator and title created.
  Value: atom, list

buttons
  Specifies a list of compound strings which are used as the labels of the
  buttons created. This list contains one element for each button, separator
  and title created.
  Value: atom, list

buttonSet
  Specifies which button of a pull-down submenu in a RadioBox or an OptionMenu
  is active. The value specifies the ordinal number of a ToggleButtonGadget
  from a RadioBox, or of a PushButtonGadget in a submenu of an Option menu,
  which has been activated. The value 0 represents the first button.
  Value: integer

buttonType
  Specifies a list of button types for the buttons which are to be created. This
  list contains one element for each button, separator and title created. If this
  resource is not set, then in a menu bar every button will be a CascadeButton
  gadget, and in any other type of RowColumn widget every button will be a
  PushButton gadget.
  Value:

  cascadebutton
    CascadeButton gadget for a menu bar, a pop-up menu, a pull-down
    menu or a pull-down submenu
Widgets

checkbutton

ToggleButton gadget for a confirm box, a pop-up menu or a pull-down menu.

double_separator

Separator gadget for a menu bar, a pop-up menu, a pull-down menu or a pull-down submenu. The type of the separator is double_line.

pushbutton

Push button for a menu bar, a pop-up menu, a pull-down menu or a pull-down submenu.

radiobutton

ToggleButton gadget for a radio box, a pop-up menu or a pull-down menu.

separator

Separator gadget for a pop-up menu, a pull-down menu or a pull-down submenu.

title

Label gadget, used as the title for a pop-up menu or a pull-down menu.

optionLabel

Specifies a compound string for the label string, which appears on the left-hand side of an option menu.

Value: atom, list

optionMnemonic

Specifies the key symbol (KeySym) of a key which, when it is pressed at the same time as the Alt key, results in the output of the pull-down menu associated with an option menu.

Value: atom

postFromButton

For a pull-down menu, this defines a button in the associated parent widget to which the submenu is assigned. The submenu can then be posted using this button. The value identifies the nth cascade button, or the nth CascadeButton gadget, which belongs to the pull-down menu’s parent widget. The value 0 represents the first button.

Value: integer

simpleCallback

Specifies a callback predicate which will be called when a button is activated or its value changes. This callback predicate is recorded for each button when it is created. For a CascadeButton or PushButton gadget, the callback predicate will be added to the activateCallback, and it will be called if the button is activated. For a ToggleButton gadget, the callback predicate will be added to the valueChangedCallback and will be called if the value of the
button changes. The ordinal number of the button will be passed in the
ClientData argument.
Value: can only be set by xtAddCallback/4

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entryCallback</td>
<td>activate</td>
<td>Child button activated</td>
</tr>
<tr>
<td>mapCallback</td>
<td>map</td>
<td>Mapping of the window associated with the RowColumn widget has been initiated</td>
</tr>
<tr>
<td>unmapCallback</td>
<td>unmap</td>
<td>Mapping of the window associated with the RowColumn widget has been canceled</td>
</tr>
<tr>
<td>tearOffMenuActivateCallback</td>
<td>tear_off_activate</td>
<td>tear-off MenuPane is about to be activated</td>
</tr>
<tr>
<td>tearOffMenuDeactivateCallback</td>
<td>tear_off_deactivate</td>
<td>tear-off MenuPane is about to be deactivated</td>
</tr>
</tbody>
</table>

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>atom</td>
<td>activate</td>
<td>Activated child widget</td>
</tr>
<tr>
<td>callbackstruct</td>
<td>list</td>
<td>activate</td>
<td>Callback components of the initiating callback in the child widget</td>
</tr>
</tbody>
</table>

See also

BulletinBoard, Frame, ScrolledWindow, PanedWindow
Widgets

Scale

The Scale widget is used to display a scale in which a value within a value range can be indicated and changed. The value is indicated in the Scale widget by means of a slider.

Class xmScaleWidgetClass
Superclasses Manager, Constraint, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimalPoints</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>dragCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>fontList</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>highlightOnEnter</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>highlightThickness</td>
<td>integer</td>
<td>CSG</td>
<td>2</td>
</tr>
<tr>
<td>maximum</td>
<td>integer</td>
<td>CSG</td>
<td>100</td>
</tr>
<tr>
<td>minimum</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>orientation</td>
<td>atom</td>
<td>CSG</td>
<td>vertical</td>
</tr>
<tr>
<td>processingDirection</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>scaleHeight</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>scaleMultiple</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>scaleWidth</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>showValue</td>
<td>atom, list</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>titleString</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>traversalOn</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>value</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
</tbody>
</table>

decimalPoints

Specifies the number of decimal places to be used in calculating the value displayed by the slider. Example: A slider value (resource value) of 2,350 and a value of 2 for decimalPoints produces a display value of 23.50.

Value: integer

dragCallback

Specifies the callback list which will be called if the slider is moved.

Value: can only be set by xtAddCallback/4

fontList

Specifies the font to be used for outputting the title.

Value: valid font list

highlightOnEnter

Specifies whether a rectangle is to be drawn around the widget to highlight it when the mouse pointer is moved onto it.
Scale

Value: boolean values (true, false, on, off, yes, no)

highlightThickness
Specifies the width of the highlighting rectangle, in pixels.
Value: integer

maximum
Specifies the maximum value on the scale.
Value: integer

minimum
Specifies the minimum value on the scale.
Value: integer

orientation
Defines the orientation of the scale.
Value: vertical or horizontal

processingDirection
Specifies whether the values on the scale are arranged in ascending or descending order. This resource defines where the maximum value (value of the maximum resource) is shown relative to the minimum value (value of the minimum resource).
Value: max_on_top at the top of a vertical scale
max_on_bottom at the bottom of a vertical scale
max_on_left on the left of a horizontal scale
max_on_right on the right of a horizontal scale

scaleHeight
Defines the height in pixels of the slider area.
Value: integer

scaleMultiple
Determines the size of the increments by which the slider moves along the scale when the user prompts a movement. The increment is the difference between the maximum and minimum resources, divided by this value.
Value: integer

scaleWidth
Defines the width in pixels of the slider area.
Value: integer

showValue
Specifies whether the current value of the slider setting is to be displayed as a number or not.
Value: boolean values (true, false, on, off, yes, no)
Widgets

Scale

titleString
  Specifies a text for the title of the widget.
  Value: valid XmString

traversalOn
  Switches traversal on and off. The value true for traversal means that control via the keyboard is permissible. The key bindings (translations) are dependent on the widget class which is being used.
  Value: boolean values (true, false, on, off, yes, no)

value
  Specifies the current value for the position of the slider on the scale between maximum and minimum.
  Value: number

valueChangedCallback
  Specifies the callback list which will be called if the value is changed.
  Value: can only be set by xtAddCallback/4

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dragCallback</td>
<td>drag</td>
<td>The slider has been dragged</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>value_changed</td>
<td>The value of the slider has changed</td>
</tr>
</tbody>
</table>

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>integer</td>
<td>drag</td>
<td>Slider position</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value_changed</td>
<td></td>
</tr>
</tbody>
</table>
**Screen**

The Screen object is used to store information specific to a screen. It is created automatically when the user calls `xtInitialize/5/6`. Initial values for the resources can only be set in resource files.

Class: `xmScreenWidgetClass`  
Superclasses: Core

### Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>darkThreshold</td>
<td>integer</td>
<td>C</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>defaultCopyCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultValidCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultInvalidCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultLinkCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultMoveCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultNoneCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultSourceCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>defaultValidCursorIcon</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>font</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>foregroundThreshold</td>
<td>integer</td>
<td>C</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>horizontalFontUnit</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>lightThreshold</td>
<td>integer</td>
<td>C</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>menuCursor</td>
<td>atom</td>
<td>C</td>
<td>arrow</td>
</tr>
<tr>
<td>moveOpaque</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>unpostBehavior</td>
<td>atom</td>
<td>CSG</td>
<td>unpost_and_replay</td>
</tr>
<tr>
<td>verticalFontUnit</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

**darkThreshold**  
Specify level of perceived brightness for a color.  
Value: integer ranging from 0 to 100

**defaultCopyCursorIcon**  
Specifies DragIcon for copy operation.  
Value: widget

**defaultInvalidCursorIcon**  
Specifies DragIcon for invalid drop site.  
Value: widget

**defaultLinkCursorIcon**  
Specifies DragIcon for link operation.  
Value: widget
defaultMoveCursorIcon
  Specifies DragIcon for move operation.
  Value: widget

defaultNoneCursorIcon
  Specifies DragIcon used when cursor is not over a drop site.
  Value: widget

defaultSourceCursorIcon
  Specifies DragIcon used when no source cursor icon is provided by a Drag-
  Context.
  Value: widget

defaultValidCursorIcon
  Specifies DragIcon used when cursor is over a valid drop site.
  Value: widget

font
  Specifies font to be used for computing font units.

foregroundThreshold
  Specify level of perceived brightness for a color.
  Value: integer ranging from 0 to 100

horizontalFontUnit
  Specifies horizontal component of font units.
  Value: integer

lightThreshold
  Specify level of perceived brightness for a color.
  Value: integer ranging from 0 to 100

menuCursor
  Specifies cursor used for menus.
  Value: valid cursor

moveOpaque
  If true, use representation of window when moving a window. If false, use
  just outline of window.
  Value: boolean values (true, false, on, off, yes, no)

unpostBehavior
  Specify whether to replay a button event to the window that contains the
  pointer when a menu is unposted.
  Value: unpost_and_replay, replay button event, unpost, do not replay
verticalFontUnit
   Specifies vertical component of font units.
   Value: integer

Compatibility

   Screen is not available in the interface to OSF/Motif 1.1.

See also

   Display, xmGetXmScreen/1
ScrollBar (widget)

The ScrollBar widget is used to move data within a window when there are more items of data than can be shown at the same time in the window area.

Class xmScrollBarWidgetClass
Superclasses Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>decrementCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>dragCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>increment</td>
<td>integer</td>
<td>CSG</td>
<td>1</td>
</tr>
<tr>
<td>incrementCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>initialDelay</td>
<td>integer</td>
<td>CSG</td>
<td>250</td>
</tr>
<tr>
<td>maximum</td>
<td>integer</td>
<td>CSG</td>
<td>100</td>
</tr>
<tr>
<td>minimum</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>orientation</td>
<td>atom</td>
<td>CSG</td>
<td>vertical</td>
</tr>
<tr>
<td>pageDecrementCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>pageIncrement</td>
<td>integer</td>
<td>CSG</td>
<td>10</td>
</tr>
<tr>
<td>pageIncrementCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>processingDirection</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>repeatDelay</td>
<td>integer</td>
<td>CSG</td>
<td>50</td>
</tr>
<tr>
<td>showArrows</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>sliderSize</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>toBottomCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>toTopCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>troughColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>value</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
</tbody>
</table>

decrementCallback
Specifies the callback list which will be called if the value is decremented.
Value: can only be set by xtAddCallback/4

dragCallback
Specifies the callback list which will be called if the slider is moved.
Value: can only be set by xtAddCallback/4

increment
Specifies the amount by which the value will be incremented or decremented when the user moves the slider by one increment.
Value: integer
ScrollBar Widgets

incrementCallback
Specifies the callback list which will be called if the value is incremented.
Value: can only be set by xtAddCallback/4

initialDelay
Specifies the delay in milliseconds from the time that the user presses a mouse button within the scroll bar before the slider moves continuously.
Value: integer

maximum
Specifies the maximum value of the scroll bar.
Value: integer

minimum
Specifies the minimum value of the scroll bar.
Value: integer

orientation
Defines the orientation of the scroll bar.
Value: vertical or horizontal

pageDecrementCallback
Specifies the callback list which will be called if a backward page feed is initiated.
Value: can only be set by xtAddCallback/4

pageIncrement
Specifies the amount by which the value will be incremented or decremented if the user initiates a page feed.
Value: integer

pageIncrementCallback
Specifies the callback list which will be called if a forward page feed is initiated.
Value: can only be set by xtAddCallback/4

processingDirection
Specifies how the scroll bar is oriented. This resource defines where the maximum value (value of the maximum resource) is shown relative to the minimum value (value of the minimum resource).

Value: max_on_top At the top of a vertical scroll bar
max_on_bottom At the bottom of a vertical scroll bar
max_on_left On the left of a horizontal scroll bar
max_on_right On the right of a horizontal scroll bar
Widgets

ScrollBar

repeatDelay

Defines the time to wait, in milliseconds, between two slider movements after the time specified by the initialDelay resource has elapsed.

Value: integer

showArrows

Specifies whether arrows are to be displayed.

Value: boolean values (true, false, on, off, yes, no)

sliderSize

Defines the size of the slider in pixels.

Value: integer

toBottomCallback

Specifies the callback list which will be called if the cursor is positioned at the end of the scroll bar.

Value: can only be set by xtAddCallback/4

toTopCallback

Specifies the callback list which will be called if the cursor is positioned at the start of the scroll bar.

Value: can only be set by xtAddCallback/4

troughColor

Specifies the color of the slider guide.

Value: valid color

toBottomCallback

Specifies the current value for the position of the slider along the scroll bar between maximum and minimum.

Value: integer

valueChangedCallback

Specifies the callback list which will be called if the value is changed.

Value: can only be set by xtAddCallback/4
### Callbacks

<table>
<thead>
<tr>
<th>Callback name</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decrementCallback</td>
<td>decrement</td>
<td>Decrement arrow activated</td>
</tr>
<tr>
<td>dragCallback</td>
<td>drag</td>
<td>Slider has been dragged</td>
</tr>
<tr>
<td>incrementCallback</td>
<td>increment</td>
<td>Increment arrow activated</td>
</tr>
<tr>
<td>pageDecrementCallback</td>
<td>page_decrement</td>
<td>Decrement slider activated for page movement</td>
</tr>
<tr>
<td>pageIncrementCallback</td>
<td>page_increment</td>
<td>Increment slider activated for page movement</td>
</tr>
<tr>
<td>toBottomCallback</td>
<td>to_bottom</td>
<td>Value changed to minimum</td>
</tr>
<tr>
<td>toTopCallback</td>
<td>to_top</td>
<td>Value changed to maximum</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>value_changed</td>
<td>Value of the slider has changed</td>
</tr>
</tbody>
</table>

### Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>integer</td>
<td>all</td>
<td>Current slider position</td>
</tr>
<tr>
<td>pixel</td>
<td>integer</td>
<td>to_top</td>
<td>Mouse position when selected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to_bottom</td>
<td>Mouse position when selected</td>
</tr>
</tbody>
</table>
ScrolledWindow

The ScrolledWindow widget is a window which realizes a section of a larger widget which lies behind it. This section can be moved with the help of the scroll bar.

Class: xmScrolledWindowWidgetClass
Superclasses: Manager, Constraint, Composite

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>clipWindow</td>
<td>atom</td>
<td>G</td>
<td>none</td>
</tr>
<tr>
<td>horizontalScrollBar</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>scrollBarDisplayPolicy</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>scrollBarPlacement</td>
<td>atom</td>
<td>CSG</td>
<td>bottom_right</td>
</tr>
<tr>
<td>scrolledWindowMarginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>scrolledWindowMarginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>scrollingPolicy</td>
<td>atom</td>
<td>CG</td>
<td>application_defined</td>
</tr>
<tr>
<td>spacing</td>
<td>integer</td>
<td>CSG</td>
<td>4</td>
</tr>
<tr>
<td>traverseObscuredCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>verticalScrollBar</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>visualPolicy</td>
<td>atom</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>workWindow</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
</tbody>
</table>

clipWindow

Specifies the widget which is to be used for the area to be displayed.
Value: name of the widget

horizontalScrollBar

Specifies the widget which is to be used as the horizontal scroll bar.
Value: name of the widget

scrollBarDisplayPolicy

Specifies how the scroll bar is to be displayed.
Value: as_needed
A scroll bar will be displayed only if the work area is larger than the area to be displayed.
static
A scroll bar is always displayed.

scrollBarPlacement

Defines the position of the scroll bars.
Value: top_left
bottom_left
top_right
bottom_right
scrolledWindowMarginHeight
Defines the height, in pixels, of the top and bottom borders.
Value: integer

scrolledWindowMarginWidth
Defines the width, in pixels, of the left- and right-hand borders.
Value: integer

scrollingPolicy
Specifies how the movements will be made.
Value:

  automatic
    Scroll bars will be created automatically and provided with callbacks.
    The value of the visualPolicy resource is set to constant.
  application_defined
    The creation of the scroll bars and the actual movements will be han-
    dled by the application.

spacing
Specifies the distance, in pixels, between the scroll bar and the work area.
Value: integer

traverseObscuredCallback
Specifies the list of callbacks that is called when traversing to a widget within
the scrolled window that is obscured.
Value: can only be set by xtAddCallback/4

verticalScrollBar
Specifies the widget which is to be used as the vertical scroll bar.
Value: name of the widget

visualPolicy
Specifies whether the ScrolledWindow widget may automatically expand to
the full size of the work area.
Value:

  variable
    The widget will automatically be adjusted to the size of the work area.
    As a result, any change in the size of the work area will influence the
    ScrolledWindow widget. The scrollBarDisplayPolicy resource will be
given the value 'static', i.e. the scroll bars will always be visible.
  constant
    The widget will not be adjusted automatically.
workWindow

Specifies the widget which is to be used as the work area.
Value: atom, name of the widget

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>traverseObscuredCallback</td>
<td>obscured_traversal</td>
<td>Traversing to obscured widget</td>
</tr>
</tbody>
</table>

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>traversal_destination</td>
<td>atom</td>
<td>obscured_traversal</td>
<td>Widget to traverse to</td>
</tr>
<tr>
<td>traversal_direction</td>
<td>atom</td>
<td>obscured_traversal</td>
<td>see \texttt{xmProcessTraversal/2}</td>
</tr>
</tbody>
</table>

See also

PanedWindow
**SelectionBox**

The SelectionBox widget is a composite widget which is used to select an entry from a list of entries. This widget consists of a List widget and a Text widget, into which a marked entry is copied. The display will show three or four push buttons which can be given callbacks.

- **OK**: The value shown in the Text widget will be accepted, and the SelectionBox widget will be closed.
- **Apply**: The value shown in the Text widget will be accepted, the SelectionBox widget will not be closed.
- **Cancel**: The SelectionBox widget will be closed.
- **Help**: Help information will be offered.

**Class**  
`xmSelectionBoxWidgetClass`

**Superclasses**  
BulletinBoard, Manager, Constraint, Composite, Core

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>applyCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>applyLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Apply'</td>
</tr>
<tr>
<td>cancelCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>cancelLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Cancel'</td>
</tr>
<tr>
<td>childPlacement</td>
<td>atom, list</td>
<td>CSG</td>
<td>place_above_selection</td>
</tr>
<tr>
<td>dialogType</td>
<td>atom</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>helpLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Help'</td>
</tr>
<tr>
<td>listItemCount</td>
<td>integer</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>listItems</td>
<td>list</td>
<td>CSG</td>
<td>[]</td>
</tr>
<tr>
<td>listLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>listVisibleItemCount</td>
<td>integer</td>
<td>CSG</td>
<td>8</td>
</tr>
<tr>
<td>minimizeButtons</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>mustMatch</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>nomatchCallback</td>
<td>atom, list</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>okCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>okLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'OK'</td>
</tr>
<tr>
<td>selectionLabelString</td>
<td>atom, list</td>
<td>CSG</td>
<td>'Selection'</td>
</tr>
<tr>
<td>textAccelerators</td>
<td>atom</td>
<td>CSG</td>
<td>(default)</td>
</tr>
<tr>
<td>textColumns</td>
<td>integer</td>
<td>CSG</td>
<td>20</td>
</tr>
<tr>
<td>textString</td>
<td>atom, list</td>
<td>CSG</td>
<td>''</td>
</tr>
</tbody>
</table>

**applyCallback**  
Specifies the callback list which will be called if the Apply button is activated.

Value: can only be set by `xtAddCallback/4`
Widgets

SelectionBox

applyLabelString

Defines the text for the Apply button.
Value: valid XmString

cancelCallback

Specifies the callback list which will be called if the Cancel button is activated.
Value: can only be set by xtAddCallback/4

cancelLabelString

Defines the text for the Cancel button.
Value: valid XmString

childPlacement

Specifies the placement of the work area child.
Value: place_above_selection, place_below_selection, or place_top.

dialogType

Specifies which child widgets of the SelectionBox widget are initially to be created and managed. The possible child widgets are: a List widget, a Text widget, and four PushButton widgets
Value:

dialog_prompt

There is no List widget, no associated label and no Apply button.
dialog_selection

All the child widgets will be created and managed.
dialog_work_area

The Apply button will not be managed.
dialog_file_selection

All child widgets will be created and managed.

helpLabelString

Defines the text for the Help button.
Value: valid XmString

listItemCount

Contains the number of entries in the list. The value is set automatically to the length of the listItemCount resource list.
Value: integer

listItems

Contains the list entries.
Value: list of XmStrings
SelectionBox

listLabelString
    Defines the text which is output above the list.
    Value: valid XmString

listVisibleItemCount
    Specifies the number of entries which can be seen.
    Value: integer

minimizeButtons
    Specifies whether the height and width of each button will be made the same as the largest one.
    Value: boolean values (true, false, on, off, yes, no)

mustMatch
    Specifies whether the text input by the user will be checked or not.
    Value: boolean values (true, false, on, off, yes, no)

noMatchCallback
    Specifies the callback list which will be called if the user's input fails the check.
    Value: can only be set by xtAddCallback/4

okCallback
    Specifies the callback list which will be called if the OK button is activated.
    Value: can only be set by xtAddCallback/4

okLabelString
    Defines the text for the OK button.
    Value: valid XmString

selectionLabelString
    Defines the text which is output above the text field.
    Value: valid XmString

textAccelerators
    Specifies the accelerator table which can be used to position onto elements in the Text widget.
    Value: valid accelerator table

textColumns
    Specifies the number of columns in the Text widget.
    Value: integer

textString
    Contains the text which is entered in the Text widget.
    Value: valid XmString
## Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applyCallback</td>
<td>apply</td>
<td>Apply button activated</td>
</tr>
<tr>
<td>cancelCallback</td>
<td>cancel</td>
<td>Cancel button activated</td>
</tr>
<tr>
<td>noMatchCallback</td>
<td>no_match</td>
<td>There is no match with the user’s input</td>
</tr>
<tr>
<td>okCallback</td>
<td>ok</td>
<td>OK button activated</td>
</tr>
</tbody>
</table>

## Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valueString</td>
<td>list</td>
<td>all</td>
<td>Selected value</td>
</tr>
</tbody>
</table>

See also

- MessageBox
- FileSelectionBox
Separator (widget)

The Separator widget draws a line to separate different components of a window from one another.

Class: xmSeparatorWidgetClass
Superclasses: Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>orientation</td>
<td>atom</td>
<td>CSG</td>
<td>horizontal</td>
</tr>
<tr>
<td>separatorType</td>
<td>atom</td>
<td>CSG</td>
<td>shadow_etched_in</td>
</tr>
</tbody>
</table>

margin

Specifies the distance, in pixels, between the border of the separator and the line which is drawn.

Value: integer

orientation

Defines the orientation of the line.

Value: vertical or horizontal

separatorType

Specifies the type of line.

Value:
- single_line
- single_dashed_line
- shadow_etched_in
- no_line
- double_line
- double_dashed_line
- shadow_etched_out
Separator (gadget)

The Separator gadget corresponds to the Separator widget.

Class xmSeparatorGadgetClass
Superclasses Gadget, RectObj, Object

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.
Shell

The Shell class of widgets cannot itself be instantiated. It is only implemented as a common superclass for all shell widget classes.

Class shellWidgetClass
Superclasses Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowShellResize</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>createPopupChildProc</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>geometry</td>
<td>atom</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>overrideRedirect</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>popdownCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>popupCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>saveUnder</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>visual</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

allowShellResize
Specifies whether the shell for the child widget permits changes in position and size.
Value: boolean values (true, false, on, off, yes, no)

createPopupChildProc
Specifies which procedure is to be executed if the widget is called by xtPopup/2.
Value: cannot be used in IF/Prolog

gallery
Specifies the geometry values for the widget, namely the X coordinate, Y coordinate, width and height. The value of this resource will be queried only if the widget has not been realized and the number of its managed children has changed. geometry is specified as follows:

\[
\text{width} \times \text{height} + xoff + yoff
\]

width is the width in pixels or characters.
height is the height in pixels or characters.
+xoff specifies the distance, in pixels, from the left-hand border of the window to the left-hand border of the screen.
-xoff specifies the distance, in pixels, from the right-hand border of the window to the right-hand border of the screen.
Widgets

+yoff specifies the distance, in pixels, from the top border of the window to the top border of the screen.

-yoff specifies the distance, in pixels, from the bottom border of the window to the bottom border of the screen.
Value: valid geometric specification

overrideRedirect
Specifies whether the widget is a temporary window which is to be ignored by the Window Manager.
Value: boolean values (true, false, on, off, yes, no)

popupCallback
Specifies the callback list which will be called if xtPopdown/1 is executed.
Value: can only be set by xtAddCallback/4

popupCallback
Specifies the callback list which will be called if xtPopup/2 is executed.
Value: can only be set by xtAddCallback/4

saveUnder
Specifies whether concealed areas of the window are to be saved or not.
Value: boolean values (true, false, on, off, yes, no)

visual
Specifies the visual to be used when the widget is created.
Value: cannot be used in IF/Prolog

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>popupCallback</td>
<td></td>
<td>Called during xtPopup/2</td>
</tr>
<tr>
<td>popdownCallback</td>
<td></td>
<td>Called during xtPopdown/1</td>
</tr>
</tbody>
</table>
Text and TextField

The Text and TextField classes of widget can be used to create windows into which text can be input.

The Text widget can be used to create a window in which one or more lines of text can be input and edited.

Class xmTextWidgetClass
Superclasses Primitive, Core

A TextField widget is a Text widget which has only a single line. Its functionality is broadly the same as that of the Text widget.

Class xmTextFieldWidgetClass
Superclasses Primitive, Core

Some of the resources of the Text widget are inapplicable for TextField, as they are meaningless for single-line text items. In the table below, the resources which do not apply for TextField widgets are marked with an asterisk (*).

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>* autoShowCursorPosition</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>cursorPosition</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>editable</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>* editMode</td>
<td>atom</td>
<td>CSG</td>
<td>single_line_edit</td>
</tr>
<tr>
<td>focusCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>gainPrimaryCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>losePrimaryCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>losingFocusCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>marginHeight</td>
<td>integer</td>
<td>CSG</td>
<td>5</td>
</tr>
<tr>
<td>marginWidth</td>
<td>integer</td>
<td>CSG</td>
<td>5</td>
</tr>
<tr>
<td>maxLength</td>
<td>integer</td>
<td>CSG</td>
<td>(largest integer value)</td>
</tr>
<tr>
<td>modifyVerifyCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>motionVerifyCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>* source</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>* topCharacter</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>value</td>
<td>atom, list</td>
<td>CSG</td>
<td>' '</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>verifyBell</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
</tbody>
</table>

Resources for text input

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pendingDelete</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
</tbody>
</table>

IF/Prolog V5.2 147 Motif Manual
Widgets

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectionArray</td>
<td>list</td>
<td>CSG</td>
<td>(default)</td>
</tr>
<tr>
<td>selectionArrayCount</td>
<td>integer</td>
<td>G</td>
<td>(default)</td>
</tr>
<tr>
<td>selectThreshold</td>
<td>integer</td>
<td>CSG</td>
<td>5</td>
</tr>
</tbody>
</table>

Resources for text output

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>blinkRate</td>
<td>integer</td>
<td>CSG</td>
<td>500</td>
</tr>
<tr>
<td>columns</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>cursorPositionVisible</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>fontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>* resizeHeight</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>resizeWidth</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>* rows</td>
<td>integer</td>
<td>CSG</td>
<td>(default)</td>
</tr>
<tr>
<td>* wordWrap</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
</tbody>
</table>

Resources which are used when the parent widget is a ScrolledWindow

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>* scrollHorizontal</td>
<td>atom</td>
<td>CG</td>
<td>true</td>
</tr>
<tr>
<td>* scrollLeftSide</td>
<td>atom</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>* scrollTopSide</td>
<td>atom</td>
<td>CG</td>
<td>false</td>
</tr>
<tr>
<td>* scrollVertical</td>
<td>atom</td>
<td>CG</td>
<td>true</td>
</tr>
</tbody>
</table>

activateCallback

Specifies the callback list which will be called if an entry is activated.
Value: can only be set by xtAddCallback/4

autoShowCursorPosition

Specifies whether the insertion cursor is always to be within the visible text. If this resource is set to true, the text will automatically be moved whenever this cursor is outside the visible text.
Value: boolean values (true, false, on, off, yes, no)

cursorPosition

Specifies the position of the cursor used for text insertion. This position is
declared by the number of characters from the start of the text. The first
character is in position number 0.
Value: integer

editable

Specifies whether the user may edit the text.
Value: boolean values (true, false, on, off, yes, no)
editMode
  Specifies whether the widget is to have one or more lines.
  Value: single_line_edit  one line
         multi_line_edit  several lines

focusCallback
  Specifies the callback list which will be called if the widget is given the input focus.
  Value: can only be set by xtAddCallback/4

gainPrimaryCallback
  Specifies the callback list which will be called if the widget receives the first selection.
  Value: can only be set by xtAddCallback/4

losePrimaryCallback
  Specifies the callback list which will be called if the widget loses the first selection.
  Value: can only be set by xtAddCallback/4

losingFocusCallback
  Specifies the callback list which will be called if the widget loses the input focus.
  Value: can only be set by xtAddCallback/4

marginHeight
  Defines the distance, in pixels, between the top and bottom border and the text.
  Value: integer

marginWidth
  Defines the distance, in pixels, between the right and left border and the text.
  Value: integer

maxLength
  Specifies the maximum length of the character string entered.
  Value: integer

modifyVerifyCallback
  Specifies the callback list which will be called if text is deleted or inserted by means of the keyboard.
  Value: can only be set by xtAddCallback/4
motionVerifyCallback
Specifies the callback list which will be called if the input cursor is moved to a new position.
Value: can only be set by xtAddCallback/4

source
Specifies a text source which a number of widgets can share. In IF/Prolog, this cannot be specified by a resource. The predicate xmTextSetSource/4 enables a corresponding function to be performed.
Value: cannot be used from within IF/Prolog

topCharacter
Specifies the position of the first character displayed.
Value: integer

value
Contains the edited character string.
Value: atom, or a list of atoms (lines)

valueChangedCallback
Specifies the callback list which will be called if the entry in the widget is changed.
Value: can only be set by xtAddCallback/4

verifyBell
Specifies whether an audible signal will be output if the callback lists modifyVerifyCallback and motionVerifyCallback have been completed and there is no further action.
Value: boolean values (true, false, on, off, yes, no)

Resources for text input

pendingDelete
Specifies whether to delete marked text when new text is inserted.
Value: boolean values (true, false, on, off, yes, no)

selectionArray
Specifies the functions which are initiated by multiple clicking. These functions are held as constants in an array. After the first click of a mouse button, each subsequent click made within one half second will increment the counter in the array by one step, and will execute the associated function.
Value: array containing the elements:
select_position
   The first click of the mouse sets the position of the input cursor

select_word
   The second click of the mouse selects a word

select_line
   The third click of the mouse selects a line

select_all
   The fourth click of the mouse selects everything (does not apply in the case of TextField)

selectionArrayCount
   Specifies the number of elements in the selectionArray array. The value is set automatically to the length of the selectionArray resource list.
   Value: integer

selectThreshold
   Specifies the number of pixels by which the mouse pointer must be moved from a given position in order to mark the next character.
   Value: integer

**Resources for text output**

blinkRate
   Specifies the blink rate of the cursor in milliseconds, i.e. the time between two successive blinks.
   Value: integer

columns
   Specifies the number of text columns.
   Value: integer

cursorPositionVisible
   Specifies whether the cursor is visible or not.
   Value: boolean values (true, false, on, off, yes, no)

fontList
   Specifies the font to be used for output of text.
   Value: valid font list

resizeHeight
   Specifies whether or not the height of the text is adjusted to the overall text in the widget. This resource will be ignored if the scrollVertical resource is set to true.
   Value: boolean values (true, false, on, off, yes, no)
Widgets Text, TextField

**resizeWidth**

Specifies whether or not the width of the text is adjusted to the overall text in the widget. This resource will be ignored if the wordWrap resource is set to `true`.

Value: boolean values (true, false, on, off, yes, no)

**rows**

Specifies the number of rows which are initially to be created. This resource is only meaningful if the value of the editMode resource is set to `multiline_edit`.

Value: integer

**wordWrap**

Specifies whether or not line breaks are inserted.

Value: boolean values (true, false, on, off, yes, no)

**Resources which are used if the superior widget is a ScrolledWindow widget**

**scrollHorizontal**

Specifies whether a horizontal scroll bar is added.

Value: boolean values (true, false, on, off, yes, no)

**scrollLeftSide**

Specifies that the vertical scroll bar is to be shown on the left-hand side of the text.

Value: boolean values (true, false, on, off, yes, no)

**scrollTopSide**

Specifies that the horizontal scroll bar is to be shown at the top.

Value: boolean values (true, false, on, off, yes, no)

**scrollVertical**

Specifies whether a vertical scroll bar is to be shown.

Value: boolean values (true, false, on, off, yes, no)

**Callbacks**

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCallback</td>
<td>activate</td>
<td>Widget activated</td>
</tr>
<tr>
<td>focusCallback</td>
<td>focus</td>
<td>Input focus received</td>
</tr>
<tr>
<td>losingFocusCallback</td>
<td>losing_focus</td>
<td>Input focus lost</td>
</tr>
<tr>
<td>modifyVerifyCallback</td>
<td>modifying_text_value</td>
<td>Process before text is inserted or deleted</td>
</tr>
<tr>
<td>motionVerifyCallback</td>
<td>moving_text_insert</td>
<td>Process before the cursor is moved</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>value_changed</td>
<td>Text deleted or inserted</td>
</tr>
</tbody>
</table>

Motif Manual 152 IF/Prolog V5.2
### Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doit</td>
<td>atom</td>
<td>losing_focusmodifying_text_value moving_insert_cursor</td>
<td>Specifies which action which triggered the callback is executed.</td>
</tr>
<tr>
<td>currinsert</td>
<td>integer</td>
<td>losing_focusmodifying_text_value moving_insert_cursor</td>
<td>Current position of the insertion cursor.</td>
</tr>
<tr>
<td>newinsert</td>
<td>integer</td>
<td>losing_focusmodifying_text_value moving_insert_cursor</td>
<td>New position of the insertion cursor.</td>
</tr>
<tr>
<td>startpos</td>
<td>integer</td>
<td>losing_focus</td>
<td>modifying_text_value</td>
</tr>
<tr>
<td>endpos</td>
<td>integer</td>
<td>losing_focus</td>
<td>modifying_text_value</td>
</tr>
<tr>
<td>text</td>
<td>atom</td>
<td>modifying_text_value</td>
<td>Text to insert.</td>
</tr>
</tbody>
</table>

### Hints

You can use `xmSetCallbackcomponents/1` to set the `doit` component to `false` and thus prevent a change from being carried out. Alternatively the predicates `xmTextSetConstraints/5` and `xmTextFieldSetConstraints/5` can be used for that purpose.
ToggleButton (widget)

The ToggleButton widget is a button which can only have two different values, e.g. on and off.

Class    xmToggleButtonWidgetClass
Superclasses  Label, Primitive, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>armCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>fillOnSelect</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>indicatorOn</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>indicatorSize</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>indicatorType</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>selectColor</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>selectInsensitivePixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>selectPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>set</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>spacing</td>
<td>integer</td>
<td>CSG</td>
<td>4</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>-</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>visibleWhenOff</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
</tbody>
</table>

armCallback

Specifies the callback list which will be called if the button is marked.

Value: can only be set by xtAddCallback/4

disarmCallback

Specifies the callback list which will be called if the marking of the button is canceled.

Value: can only be set by xtAddCallback/4

fillOnSelect

Specifies whether the button indicator is to be filled with the color specified by selectColor.

Value: boolean values (true, false, on, off, yes, no)

indicatorOn

Specifies whether a toggle indicator is mapped or not.

Value: boolean values (true, false, on, off, yes, no)

indicatorSize

Specifies the dimension of the indicator.
ToggleButton Widgets

Value: integer

If the value is -1 the dimension is calculated from the size of the font which is being used.

indicatorType

Defines the type of indicator.

Value: one_of_many Diamond-shaped
       n_of_many Square

selectColor

Specifies the color to be used to fill the indicator.

Value: valid color

selectInsensitivePixmap

Specifies a pixmap to be used for representing a button which cannot be activated.

Value: pixmap (name of a bitmap file)

selectPixmap

Specifies a pixmap for a button which has been activated.

Value: pixmap (name of a bitmap file)

set

Indicates whether the button has been activated.

Value: boolean values (true, false, on, off, yes, no)

spacing

Specifies the distance between indicator and label, in pixels.

Value: integer

valueChangedCallback

Specifies the callback list which will be called if the value is changed.

Value: can only be set by xtAddCallback/4

visibleWhenOff

Specifies whether or not the button can be seen when is is not activated.

Value: boolean values (true, false, on, off, yes, no)

Callbacks

<table>
<thead>
<tr>
<th>Callbackname</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>armCallback</td>
<td>arm</td>
<td>Button marked</td>
</tr>
<tr>
<td>disarmCallback</td>
<td>disarm</td>
<td>Button marking canceled</td>
</tr>
<tr>
<td>valueChangedCallback</td>
<td>value_changed</td>
<td>Status has changed</td>
</tr>
</tbody>
</table>

IF/Prolog V5.2 155 Motif Manual
Widgets

ToggleButton

Callback Components

<table>
<thead>
<tr>
<th>Callback component</th>
<th>Data type</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>atom</td>
<td>all</td>
<td>Status</td>
</tr>
</tbody>
</table>

See also

PushButton, DrawnButton, CascadeButton
ToggleButton (gadget)

The ToggleButton gadget corresponds to the ToggleButton widget.

<table>
<thead>
<tr>
<th>Class</th>
<th>xmToggleButtonGadgetClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superclasses</td>
<td>Label, Primitive, Core</td>
</tr>
</tbody>
</table>

Hints

Gadgets cannot be controlled via translations or accelerators. They receive events from superior widgets.
**TopLevelShell**

A TopLevelShell widget is used as the top widget in a hierarchy of widgets in cases where an application requires several independent widget trees.

Class topLevelShellWidgetClass
Superclasses Shell, Composite, Core

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>iconic</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>iconName</td>
<td>atom</td>
<td>CSG</td>
<td>' '</td>
</tr>
<tr>
<td>iconNameEncoding</td>
<td>atom</td>
<td>CSG</td>
<td>'STRING'</td>
</tr>
</tbody>
</table>

**iconic**

Specifies whether the widget is to appear as a normal window or as an icon after it has been realized. The value `true` means that the window will appear as an icon.

Value: boolean values (true, false, on, off, yes, no)

**iconName**

Defines the name to be given to the window if it is iconified.

Value: name of the icon

**iconNameEncoding**

Specifies the type of the property which supplies the encoding of the iconName string.

Value: type of the property
**TransientShell**

TransientShell widgets are used for shell widgets which are managed by the Window Manager but cannot be separately iconified, such as dialog boxes, for example.

Class transientShellWidgetClass
Superclasses VendorShell, WMShell, Shell, Composite, Core

**Resources**

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>transientFor</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
</tbody>
</table>

transientFor  
Specifies a widget for which the widget functions as a pop-up widget.  
Value: name of a widget

**See also**

TransientShell, ApplicationShell
VendorShell

The VendorShell class of widgets can be used as a superclass for all Shell classes of widget. In addition to the specifications of the WMShell widget class, VendorShell contains further information about the appearance and behavior of the Window Manager.

Class vendorShellWidgetClass
Superclasses WMShell, Shell, Composite, Core

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>audibleWarning</td>
<td>atom</td>
<td>CSG</td>
<td>bell</td>
</tr>
<tr>
<td>buttonFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>defaultFontList</td>
<td>atom, list</td>
<td>CG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>deleteResponse</td>
<td>atom</td>
<td>CSG</td>
<td>destroy</td>
</tr>
<tr>
<td>keyboardFocusPolicy</td>
<td>atom</td>
<td>CSG</td>
<td>explicit</td>
</tr>
<tr>
<td>labelFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>mwmMenu</td>
<td>atom, list</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>shellUnitType</td>
<td>atom</td>
<td>CSG</td>
<td>pixels</td>
</tr>
<tr>
<td>textFontList</td>
<td>atom, list</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>mwmMenu</td>
<td>atom, list</td>
<td>CSG</td>
<td>''</td>
</tr>
<tr>
<td>useAsyncGeometry</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
</tbody>
</table>

audibleWarning

Determines whether an action activates its associated audible cue.
Value: bell or none

buttonFontList

Specifies the font for button descendants.
Value: valid font list

defaultFontList

Specifies the font for the child widgets.
Value: valid font list

deleteResponse

Specifies which action the Shell will execute when it receives an instruction to clear the window.
Value:
destroy

The widget will be completely destroyed, i.e. it will be removed from the screen and will no longer be managed.
unmap
The widget will be removed from the screen.

do_nothing
The instruction will be ignored.

keyboardFocusPolicy
Specifies the manner in which the widget will be given the keyboard focus.

explicit
The widget will receive the keyboard focus if it is selected by being clicked on.

pointer
The widget will receive the keyboard focus if the mouse cursor is moved onto it.

inputMethod
Specifies the string that sets the locale modifier for the input method.

Value: atom

labelFontList
Specifies the font for label descendants.

Value: valid font list

mwmDecorations
Specifies the decoration of the window frame, i.e. the design elements which are to be incorporated into the window frame. These elements are:

mwm_decor_border  2  Window border
mwm_decor_resize  4  Resize border
mwm_decor_title   8  Title line
mwm_decor_menu   16  Window menu button
mwm_decor_minimize 32  Iconifier button
mwm_decor_maximize 64  Maximize button
mwm_decor_all     1  All design elements

Value: Each of the constants differs by one bit. To set the resource, the constants for each of the required decorations should therefore be combined by an inclusive bitwise OR operation.

mwmFunctions
Defines the functions which are offered in the widget’s window menu from the button in the top left of the window frame. The following default functions are available:
mwm_func_resize 2 Enlarge or reduce the widget
mwm_func_move 4 Move the widget
mwm_func_minimize 8 Iconify the widget
mwm_func_maximize 16 Enlarge widget to fill the screen
mwm_func_close 32 Remove widget from the screen
mwm_func_all 1 All functions

Value: Each of the constants differs by one bit. To set the resource, the constants for each of the required decorations should therefore be combined by an inclusive bitwise OR operation.

mwmInputMode

Specifies the input mode.

Value: one of the following integer values

0 (mwm_input_modeless)
   No other widgets are inhibited. This is the default setting if the parent widget is a dialog shell.

1 (mwm_input_primary_application_modal)
   Inhibits inputs into ancestor widgets in the same application.

2 (mwm_input_system_modal)
   All inputs to other widgets, including those for different applications, are inhibited.

3 (mwm_input_full_application_modal)
   Any dialog with widgets in the same application is inhibited.

mwmMenu

Specifies the menu items which the Window Manager is to add at the end of the root menu.

Value: List of atoms in the form label [mnemonic][accelerator]function

treeditType

Specifies the input method style(s) available to the input manager.

Value: atom

shellUnitType

Defines a unit for enlargements, reductions and positioning.

Value: pixels Pixels
       100th_millimeters Hundredths of a millimeter
       1000th_inches Thousandths of an inch
       100th_points Hundredths of a point (1 point = 1/72 inch)
       100th_font_units Hundredths of a font unit

textFontList

Specifies the font for text and list descendants.

Value: valid font list

Motif Manual 162 IF/Prolog V5.2
useAsyncGeometry

Specifies whether the Geometry Manager will wait for the Window Manager to confirm a geometry instruction. If the value of this resource is true, the Geometry Manager will cause the waitForWm resource (WMShell widget class) to be set to false and wmTimeout to be set to 0.

Value: boolean values (true, false, on, off, yes, no)

See also

WMShell
WMShell

WMShell is a widget which defines how the Window Manager will treat the widget.

Class WMShellWidgetClass
Superclasses Shell, Composite, Core

WMShellWidgetClass defines a number of resources which are used to give the Window Manager information about the required appearance of windows and how to handle them. Further details will be found in [20] on ICCCM Inter-Client Communication Conventions.

Resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Data type</th>
<th>Access</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseHeight</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>baseWidth</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>heightInc</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>iconMask</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>iconPixmap</td>
<td>atom</td>
<td>CSG</td>
<td>none</td>
</tr>
<tr>
<td>iconWindow</td>
<td>integer</td>
<td>CSG</td>
<td>0</td>
</tr>
<tr>
<td>iconX</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>iconY</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>initialState</td>
<td>atom</td>
<td>CSG</td>
<td>normalState</td>
</tr>
<tr>
<td>input</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>maxAspectX</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>maxAspectY</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>maxHeight</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>maxWidth</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>minAspectX</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>minAspectY</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>minHeight</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>minWidth</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>title</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>titleEncoding</td>
<td>atom</td>
<td>CSG</td>
<td>string</td>
</tr>
<tr>
<td>transient</td>
<td>atom</td>
<td>CSG</td>
<td>false</td>
</tr>
<tr>
<td>waitForWm</td>
<td>atom</td>
<td>CSG</td>
<td>true</td>
</tr>
<tr>
<td>widthInc</td>
<td>integer</td>
<td>CSG</td>
<td>-1</td>
</tr>
<tr>
<td>windowGroup</td>
<td>integer</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>winGravity</td>
<td>atom</td>
<td>CSG</td>
<td>(dynamic)</td>
</tr>
<tr>
<td>wmTimeout</td>
<td>integer</td>
<td>CSG</td>
<td>5000</td>
</tr>
</tbody>
</table>
WMShell Widgets

baseHeight
Defines a base number to be used in determining a series of preferred heights for use by the Window Manager in changing the size of the widget. These preferred heights are computed as baseHeight plus a multiple of heightInc, with a minimum of minHeight and a maximum of maxHeight.

Value: integer

baseWidth
Defines a base number to be used in determining a series of preferred widths for use by the Window Manager in changing the size of the widget. These preferred widths are computed as baseWidth plus a multiple of widthInc, with a minimum of minWidth and a maximum of maxWidth.

Value: integer

heightInc
Defines the increment for size changes, in pixels.
Value: integer

iconMask
Specifies a bitmap which can be used by the Window Manager to clip the iconPixmap bitmap so that it is no longer rectangular.
Value: pixmap (name of a bitmap file)

iconPixmap
Specifies a bitmap to be used by the Window Manager when representing the application as an icon.
Value: pixmap (name of a bitmap file)

iconWindow
Specifies the window id which the Window Manager can use as an icon for the application.
Value: window id

iconX
Specifies the X coordinate of the position where the icon for the application will appear. If the Window Manager is itself responsible for managing the positions of the icons, this resource will be ignored.
Value: integer

iconY
Specifies the Y coordinate of the position where the icon for the application will appear. If the Window Manager is itself responsible for managing the positions of the icons, this resource will be ignored.
Value: integer
initialState

Defines the initial state of the widget.

Value:  
- normalState  The widget will be mapped as a normal window.
- iconicState  The widget will be mapped as an icon.

input

Specifies which input model applies for the widget and its child widgets. The meaning of the values true and false for this resource depends on whether the Window Manager controls the input focus.

<table>
<thead>
<tr>
<th>Input model</th>
<th>Input focus</th>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>No input</td>
<td>No input focus</td>
<td>false</td>
</tr>
<tr>
<td>Passive</td>
<td>No input focus</td>
<td>true</td>
</tr>
<tr>
<td>Locally active</td>
<td>Input focus</td>
<td>true</td>
</tr>
<tr>
<td>Globally active</td>
<td>Input focus</td>
<td>false</td>
</tr>
</tbody>
</table>

maxAspectX

Specifies the value of the numerator used in defining a ratio between X and Y coordinates. This numerator corresponds to the X coordinate. This and the maxAspectY resource determine maximum values for the ratio between X and Y coordinates.

Value:  integer

maxAspectY

Specifies the value of the denominator used in defining a ratio between X and Y coordinates. The denominator corresponds to the Y coordinate. This and the maxAspectX resource determine maximum values for the ratio between X and Y coordinates.

Value:  integer

maxHeight

Defines the maximum height of the window in pixels.

Value:  integer

maxWidth

Defines the maximum width of the window in pixels.

Value:  integer

minAspectX

Specifies the value of the numerator used in defining a ratio between X and Y coordinates. This numerator corresponds to the X coordinate. This and the minAspectY resource determine minimum values for the ratio between X and Y coordinates.

Value:  integer
WMShell

Widgets

minAspectY
Specifies the value of the denominator used in defining a ratio between X and Y coordinates. The denominator corresponds to the Y coordinate. This and the minAspectX resource determine minimum values for the ratio between X and Y coordinates.
Value: integer

minHeight
Defines the minimum height of the window in pixels.
Value: integer

minWidth
Defines the minimum width of the window in pixels.
Value: integer

title
Defines the title of the window.
Value: atom

titleEncoding
Specifies the type of the property which will perform the conversion of the string.
Value: atom

transient
Specifies whether the widget is given priority treatment by the Window Manager, by being immediately mapped on the screen. This applies primarily to widgets of the TransientShell class, e.g. for dialog queries.
Value: boolean values (true, false, on, off, yes, no)

waitForWm
Specifies whether the X Window System is to wait for the time specified in wmTimeout to allow the Window Manager to complete any pending tasks before assuming that there is no Window Manager.
Value: boolean values (true, false, on, off, yes, no)

widthInc
Specifies the increment, in pixels, for width modifications.
Value: integer

windowGroup
Defines the window id of the window to which the widget is to be attached. Normally, this window will represent a group of windows. The Window Manager treats all the windows which are attached to such a window in the same way, e.g. during move operations.
Value: window id
winGravity

Specifies the alignment of the window when it is positioned by the Window Manager. If no value is specified, then a value will not be determined until the widget is realized. If the value of the geometry resource (of the Shell widget class) is not empty, then its value will be used for winGravity. Otherwise, the value will be set to `north_west_gravity`.

```
forget_gravity, east_gravity,
north_west_gravity, south_west_gravity,
north_gravity, south_gravity,
north_east_gravity, south_east_gravity,
west_gravity, static_gravity, center_gravity,
```

wmTimeout

Specifies the time in milliseconds for which the X Window System will wait, if waitForWm is `true`, before assuming that there is no Window Manager.

Value: integer

See also

OverrideShell, VendorShell
Chapter 4

The Draw widget

The Draw widget makes it easier for a Prolog application to use the Xlib drawing functions. It simplifies graphics programming and opens up the possibility of using Xlib functionality in conjunction with the Prolog Motif interface.

The Draw widget is part of the Prolog OSF/Motif interface and is an additional widget which has been added to the OSF/Motif widgets.

Class xmDrawWidgetClass
Superclasses Manager, Constraint, Composite, Core

This chapter gives an overview of the functions of the Draw widget. A description of the resources, callbacks, callback components, actions and translations will be found in the section entitled Widgets and gadgets, on page 65.

Options provided by the Draw widget

- There are numerous external functions for performing a host of tasks:
  - All Xlib basic graphics elements have been implemented:
    point(s), line(s), segments, (fill-)rectangle(s), (fill-)arc(s), fill-polygon, (image)-string, copy-area, clear-area.
  - The graphics can be modified in various ways:
    line_attributes (line_width, line_style, join_style, cap_style, dashes),
    fill_params (style, rule, foreground, background, tile, stipple, arc_mode),
    font.
    In this version it is not possible to modify: function, plane_mask, subwindow_-mode, graphics-exposures, clip_origin, clip_mask.
  - There are special functions for clearing the contents of a window, for flushing the output buffer and redrawing the contents of a window.
Functions

Draw widget

- The position of the pointer can be interrogated.
- The dimensions of texts and bitmaps can be interrogated. Graphics contexts can be saved and then recalled.
- The rubber-band outlining mode can be called and terminated again.
- Using a pixmap or an operations list, the Draw widget can repaint the contents of a window.
- The Draw widget can save its contents in X11 format for window memory dumps (xwd, xwud) and can read memory dumps in this format.

- Automatic repainting of the contents of the window is effected by rerunning all saved operations that relate to the area to be redrawn.
- Up to eleven overlaid graphics contexts can be managed. The active graphics context can be saved in ten graphics context stores, and can be recalled from these.
- The appearance of graphics elements can be configured by means of resource files. There are resources for colors, bitmaps and fonts.
- Three different callback lists are available for key events, button events and motion events.
- Rubber-banding is implemented as a special inverted drawing mode (elastic outlining mode) in which the actual graphic is not stored.
- Special boolean resources control the overall behavior:
  - Compression of motion events
  - Clearing an area before it is repainted
  - Reporting of expose events during drawing
  - Automatic flushing of the buffer

Functions of the Draw widget

The functions of the Draw widget are provided in the form of Prolog predicates. All the predicates of the Draw widget have the prefix: xtDraw. The first parameter is always a widget, and this must belong to the class xmDrawWidgetClass.

This section gives a functional overview of the Draw widget predicates. Complete descriptions of the individual predicates, arranged in alphabetical order, will be found in the chapter entitled Reference section on predefined predicates (page 191).

The following description covers:

- Drawing and deleting basic elements
- Saving and recalling the graphics context
- Modifying a graphics context
- Querying the pointer position
- Setting the rubber-banding mode
- Querying the dimensions of texts and bitmaps
- Saving the window contents as a memory dump and recalling it

**Drawing and deleting basic elements**

Most of the predicates itemized below correspond to the Xlib functions with the same names (except for the initial X instead of Xt or XtDraw).

All Xlib drawing functions have additional parameters, namely the X display, a window and a graphics context. The display and the window are determined from the WidgetName parameter.

The graphics context is a special one, owned by the Draw widget. This graphics context is called the active graphics context of the Draw widget.

The active graphics context may be modified only by special predicates. There are ten graphics context stores which programmers can use to save the active graphics context and recall it. This enables them to work with different sets of graphics parameters in turn, without having to modify the active graphics context each time.

For the graphics context which is initially active, the resources 'foreground', 'background' and 'font' are set as defined by the initial resources of the widget. If these resources are modified, there will be no immediate effect on the active graphics context. The corresponding change to the initially active graphics context will only take effect when a screen is refreshed.

Following is a list of the graphic elements and the predicates which manipulate them.

<table>
<thead>
<tr>
<th>Graphic element</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>xtDrawPoint( DrawWidget, +XPosition, +YPosition)</td>
</tr>
<tr>
<td></td>
<td>xtDrawPoints( +DrawWidget, +PointList, +CoordMode)</td>
</tr>
<tr>
<td>Line</td>
<td>xtDrawLine( +DrawWidget, +XStart, +YStart, +XEnd, +YEnd)</td>
</tr>
<tr>
<td></td>
<td>xtDrawLines( +DrawWidget, +PointList, +CoordMode)</td>
</tr>
<tr>
<td></td>
<td>xtDrawDeleteLine( +DrawWidget, +XStart, +YStart, +XEnd, +YEnd)</td>
</tr>
<tr>
<td>Line segment</td>
<td>xtDrawSegments( +DrawWidget, +SegmentList)</td>
</tr>
<tr>
<td>Rectangle</td>
<td>xtDrawRectangle( +DrawWidget, +XPosition, +YPosition, +Width, +Height)</td>
</tr>
</tbody>
</table>
Graphics context

xtDrawRectangles( +DrawWidget, +RectangleList)

xtDrawFillRectangle( +DrawWidget, +XPosition, +YPosition, +Width, +Height)

xtDrawFillRectangles( +DrawWidget, +RectangleList)

Circle and ellipse

xtDrawArc( +DrawWidget, +XPosition, +YPosition, +Width, +Height, +FromAngle, +ToAngle)

xtDrawArcs( +DrawWidget, +ArcList)

xtDrawFillArc( +DrawWidget, +XPosition, +YPosition, +Width, +Height, +FromAngle, +ToAngle)

xtDrawFillArcs( +DrawWidget, +ArcList)

Text

xtDrawString( +DrawWidget, +XPosition, +YPosition, +String)

xtDrawImageString( +DrawWidget, +XPosition, +YPosition, +String)

xtDrawXmString( +DrawWidget, +XmString, +XPosition, +YPosition, +Width, +Alignment, +LayoutDirections, +ClipRectangleList)

xtDrawXmStringImage( +XmString, +XPosition, +YPosition, +Width, +Alignment, +LayoutDirections, +ClipRectangleList)

xtDrawXmStringUnderline( +DrawWidget, +XmString, +XPosition, +YPosition, +Width, +Alignment, +LayoutDirections, +ClipRectangleList, +SubXmStringToBeUnderlined)

Polygon

xtDrawFillPolygon( +DrawWidget, +PointList, +Shape, +CoordMode)

Area

xtDrawClearArea( +DrawWidget, +XPosition, +YPosition, +Width, +Height, +Exposure)

xtDrawCopyArea( +DrawWidget, +SrcXPosition, +SrcYPosition, +Width, +Height, +DestXPosition, +DestYPosition)

Bitmap

xtDrawCopyBitmap( +DrawWidget, +Bitmap, +XPosition, +YPosition)

Saving and recalling the graphics context

The predicates xtDrawStoreGC/2 and xtDrawRecallGC/2 are used, respectively, to save and recall the active graphics context. The graphics context is saved in one of ten stores for graphics contexts that are provided by the Draw widget. The second parameter specifies which store is used.
Further details about the active graphics context will be found in the section entitled *Drawing and deleting basic elements* (page 171).

**Modifying the graphics context**

The predicates listed below can be used to modify the active graphics context.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtDrawSetDashes(+DrawWidget,+Offset,+DashList)</td>
<td>Set line pattern</td>
</tr>
<tr>
<td>xtDrawSetParam(+DrawWidget,+Which,+GCValue)</td>
<td>Set parameters for graphics context</td>
</tr>
<tr>
<td>xtDrawSetTSO(Origin(+DrawWidget,+XPosition,+YPosition))</td>
<td>Set start point for pattern</td>
</tr>
</tbody>
</table>

**Querying the pointer position**

xtDrawGetXY(+DrawWidget,PointerXPosition,PointerYPosition)

and xtDrawQueryPointer(+DrawWidget,XPosition,YPosition) return the coordinates of the last pointer position.

**Setting rubber-banding mode**

The rubber-banding mode is used for representing a graphic object on the screen while it is being moved.

The elastic outline mode (rubber-banding) is a special drawing mode. Any object drawn in this mode appears inverted on the screen; as a result, a repeat of the identical drawing procedure will cancel out the first one. This mode enables temporary drawings of objects to be produced, generally the outlines of graphic elements. This makes it possible to represent the movement of graphic objects across the screen.

The pixel values for the drawing and the foreground are adjusted such that, provided they have the same values, the old pixel values of the drawing will be used. If they have different values, the result will be that the Draw widget’s foreground color is used for drawing in those areas which have the Draw widget’s background color, and vice versa. The color which will result if this operation covers a color which is neither the foreground color nor the background one is undefined. It is guaranteed that the color of the original drawing will be restored.

In the rubber-banding mode, the Draw widget does not save any drawing operations and therefore cannot reconstruct them.
Rubber-banding mode is implemented using a special graphics context as the active graphics context. This may also be modified using the functions for processing graphics contexts. However, if the foreground resource is modified, the resulting display colors will be unpredictable.

The following predicates can be used to switch the rubber-banding mode on and off.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtDrawSetRubberGC( +DrawWidget )</td>
<td>Switch on rubber-banding mode</td>
</tr>
<tr>
<td>xtDrawUnsetRubberGC( +DrawWidget )</td>
<td>Switch off rubber-banding mode</td>
</tr>
</tbody>
</table>

**Obtaining the dimensions of texts and bitmaps**

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtDrawBitmapExtents( +DrawWidget, -Width, -Height)</td>
<td>Obtain bitmap dimensions</td>
</tr>
<tr>
<td>xtDrawFontExtents( +DrawWidget, -FontDirection, -Ascent, -Descent)</td>
<td>Query size of font</td>
</tr>
<tr>
<td>xtDrawStringExtents( +DrawWidget, +String, -Width, -Ascent, -Descent)</td>
<td>Query size of character</td>
</tr>
</tbody>
</table>

**Saving and recalling window contents as a memory dump**

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtDrawQueryDumpFile( +FileName, -Width, -Height)</td>
<td>Query size of memory dump</td>
</tr>
<tr>
<td>xtDrawReadWindowDump( +DrawWidget, +SrcXPosition, +SrcYPosition, +Width, +Height, +DestXPosition, +DestYPosition, +FileName)</td>
<td>Read memory dump of a window</td>
</tr>
<tr>
<td>xtDrawWriteWindowDump( +DrawWidget, +FileName)</td>
<td>Create memory dump of window contents</td>
</tr>
</tbody>
</table>

Motif Manual 174 IF/Prolog V5.2
Chapter 5

Overview of predefined predicates by function

This section contains a summary of the predefined predicates, arranged into groups according to function.

The functor of each predicate is highlighted in boldface. Predicates for which backtracking is possible are identified by the character ‘#’ before the functor.

The set of predicates has been split into these groups:

- Callback structures
- Cascade Button
- Command
- Creation and management of widgets
- Draw
- FileSelectionBox
- Font List
- Interrogation predicates
- List
- MainWindow
- Other event sources
- Representation Types
- RowColumn
- Scroll
- ScrolledWindow
- Text

- Callbacks and actions
- Clipboard predicates
- Converter
- Drag and Drop
- Event loop
- Flush output buffer
- Interface to the Window Manager
- Keys and keyboard
- MRM predicates
- MessageBox
- Popup
- Resources
- Scale
- ScrollBar
- SelectionBox
- ToggleButton

Callback structures

**xmGetCallbackComponent/1**

Retrieve component of a callback structure
Overview

-xmGetCallbackComponents/1 Retrieve components of a callback structure
-xmSetCallbackComponents/1 Set callback components for in/out members of callback structures
-xtGetEventComponent/1 Retrieve event component
-xtGetEventComponents/1 Retrieve event components
-xtGetMultiClickTime/2 Retrieve event component

Callbacks and actions

-xtAddAction/1 Add action predicate
-xtAddActions/1 Add action predicates
-xtAddCallback/4 Add callback predicate to callback list
-xtAddCallbacks/1 Add callback predicates to callback list
-xtRemoveAllCallbacks/2 Remove all predicates from a callback list
-xtRemoveCallback/4 Remove callback
-xtRemoveCallbacks/1 Remove all callbacks

Cascade Button

-xmCascadeButtonGadgetHighlight/2 Draw highlight for the CascadeButton gadget
-xmCascadeButtonHighlight/2 Draw highlight for the CascadeButton gadget

Clipboard predicates

-xmClipboardCancelCopy/2 Cancel copying of data into clipboard
-xmClipboardCopy/5 Start copying data into the clipboard
-xmClipboardCopyByName/5 Copy data into clipboard after a delay
-xmClipboardCopyName/7 Start delayed copying of data into clipboard
-xmClipboardEndCopy/2 End copying of data into clipboard
-xmClipboardEndRetrieve/1 End copying of data out of clipboard
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmClipboardInquireCount/2</td>
<td>Query number of formats in clipboard</td>
</tr>
<tr>
<td>xmClipboardInquireFormat/3</td>
<td>Query format name for data in clipboard</td>
</tr>
<tr>
<td>xmClipboardInquireLength/3</td>
<td>Query length of data in the clipboard</td>
</tr>
<tr>
<td>xmClipboardInquirePendingItems/3</td>
<td>Query pending copy operations</td>
</tr>
<tr>
<td>xmClipboardLock/1</td>
<td>Lock clipboard</td>
</tr>
<tr>
<td>xmClipboardRegisterFormat/3</td>
<td>Register format name for clipboard</td>
</tr>
<tr>
<td>xmClipboardRetrieve/4</td>
<td>Retrieve data from clipboard</td>
</tr>
<tr>
<td>xmClipboardStartCopy/3</td>
<td>Prepare for copying data into clipboard</td>
</tr>
<tr>
<td>xmClipboardStartRetrieve/1</td>
<td>Start copying data from clipboard</td>
</tr>
<tr>
<td>xmClipboardUndoCopy/1</td>
<td>Delete last data element from clipboard</td>
</tr>
<tr>
<td>xmClipboardUnlock/2</td>
<td>Unlock clipboard</td>
</tr>
<tr>
<td>xmClipboardWithdrawFormat/2</td>
<td>Cancel outstanding data copying operations</td>
</tr>
<tr>
<td>xmGetDestination/1</td>
<td>Determine destination address for cut and paste functions</td>
</tr>
</tbody>
</table>

**Command**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmCommandAppendValue/2</td>
<td>Append text to the Command widget command</td>
</tr>
<tr>
<td>xmCommandError/2</td>
<td>Output error message to the Command widget history area</td>
</tr>
<tr>
<td>xmCommandGetChild/3</td>
<td>Query a component of the Command widget</td>
</tr>
<tr>
<td>xmCommandSetValue/2</td>
<td>Overwrite a command in the Command widget command list</td>
</tr>
</tbody>
</table>

**Converter**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmAddStringToUnitTypeConverter/0</td>
<td>Specify function for converting character strings into valid units</td>
</tr>
<tr>
<td>xmConvertUnits/6</td>
<td>Convert units</td>
</tr>
<tr>
<td>xmCreate/4</td>
<td>Convert units</td>
</tr>
<tr>
<td>xmCvtCTToXmString/2</td>
<td>Convert character string</td>
</tr>
<tr>
<td>xmCvtXmStringToCT/2</td>
<td>Convert XmString</td>
</tr>
</tbody>
</table>
Overview

xmGetPixmap/4
Generate a pixmap
xmGetPixmapByDepth/5
Generate a pixmap
xmSetFontUnit/1
Specify font size
xmSetFontUnits/2
Specify font size

Creation and management of widgets

xtCreateApplicationShell/4
Create toplevel window
xtCreateManagedWidget/5
Create and manage widget
xtCreateWidget/5
Create widget
xtDestroyWidget/1
Delete widget
xtLowerWidget/1
Place widget at bottom of window stack
xtManageChild/1
Manage child widget
xtManageChildren/1
Manage child widgets
xtMapWidget/1
Map widget
xtRaiseWidget/1
Place widget at top of window stack
xtRealizeWidget/1
Realize widget
xtRestackWidget/3
Change window stacking order
xtSetMappedWhenManaged/2
Map widget
xtSetMultiClickTime/2
Map widget
xtSetSensitive/2
Set widget so it can be activated
xtSetUserSpecifiedLocation/1
Set widget so it can be activated
xtUnmanageChild/1
Switch off management of a widget
xtUnmanageChildren/1
Switch off management of a widgets
xtUnmapWidget/1
Delete mapping of a widget
xtUnrealizeWidget/1
Delete window for a widget
xtUnregisterWidget/1
Unregister a widget

Drag and Drop

xmCreateDragIcon/4
Create a DragIcon widget
xmDragCancel/1
Abort drag operation
xmDragStart/3
Initiate a drag operation
xmDropSiteConfigureStackingOrder/3
Reorder stack of widgets registered as drop sites
### Overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmDropSiteEndUpdate</code>/1</td>
<td>Economize multiple calls to <code>xmDropSiteUpdate</code></td>
</tr>
<tr>
<td><code>xmDropSiteQueryStackingOrder</code>/3</td>
<td>Retrieve drop site parent and drop site children of widget</td>
</tr>
<tr>
<td><code>xmDropSiteRegister</code>/2</td>
<td>Register widget as a drop site</td>
</tr>
<tr>
<td><code>xmDropSiteRetrieve</code>/2</td>
<td>Retrieve resource values of a drop site</td>
</tr>
<tr>
<td><code>xmDropSiteStartUpdate</code>/1</td>
<td>Economize multiple calls to <code>xmDropSiteUpdate</code></td>
</tr>
<tr>
<td><code>xmDropSiteUnregister</code>/1</td>
<td>Free drop site information for widget</td>
</tr>
<tr>
<td><code>xmDropSiteUpdate</code>/2</td>
<td>Update resources for registered DropSite</td>
</tr>
<tr>
<td><code>xmDropTransferAdd</code>/2</td>
<td>Enable processing of additional entries after initiating a drop transfer</td>
</tr>
<tr>
<td><code>xmDropTransferStart</code>/3</td>
<td>Initiate a drop transfer</td>
</tr>
<tr>
<td><code>xmGetDragContext</code>/3</td>
<td>Return widget name of DragContext active at TimeStamp</td>
</tr>
<tr>
<td><code>xmTargetsAreCompatible</code>/2</td>
<td>Test whether targets of drop site and source are compatible</td>
</tr>
</tbody>
</table>

### Draw

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xtDrawArc</code>/7</td>
<td>Draw circular or elliptical arc</td>
</tr>
<tr>
<td><code>xtDrawArcs</code>/2</td>
<td>Draw circular or elliptical arcs</td>
</tr>
<tr>
<td><code>xtDrawBitmapExtents</code>/4</td>
<td>Obtain bitmap dimensions</td>
</tr>
<tr>
<td><code>xtDrawClear</code>/1</td>
<td>Clear contents of Draw widget window</td>
</tr>
<tr>
<td><code>xtDrawClearArea</code>/6</td>
<td>Clear window area</td>
</tr>
<tr>
<td><code>xtDrawCopyArea</code>/7</td>
<td>Copy window area</td>
</tr>
<tr>
<td><code>xtDrawCopyBitmap</code>/4</td>
<td>Copy bitmap</td>
</tr>
<tr>
<td><code>xtDrawDeleteLine</code>/5</td>
<td>Delete line</td>
</tr>
<tr>
<td><code>xtDrawFillArc</code>/7</td>
<td>Draw filled circular or elliptical arc</td>
</tr>
<tr>
<td><code>xtDrawFillArcs</code>/2</td>
<td>Draw filled circular or elliptical arcs</td>
</tr>
<tr>
<td><code>xtDrawFillPolygon</code>/4</td>
<td>Draw filled polygon</td>
</tr>
<tr>
<td><code>xtDrawFillRectangle</code>/5</td>
<td>Draw filled rectangle</td>
</tr>
<tr>
<td><code>xtDrawFillRectangles</code>/2</td>
<td>Draw filled rectangles</td>
</tr>
</tbody>
</table>

**IF/Prolog V5.2 179 Motif Manual**
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtDrawFlush/1</td>
<td>Execute output operations immediately</td>
</tr>
<tr>
<td>xtDrawFontExtents/4</td>
<td>Query size of font</td>
</tr>
<tr>
<td>xtDrawGetXY/4</td>
<td>Query last position of pointer</td>
</tr>
<tr>
<td>xtDrawImageString/4</td>
<td>Output character string with foreground and background</td>
</tr>
<tr>
<td>xtDrawLine/5</td>
<td>Draw line</td>
</tr>
<tr>
<td>xtDrawLines/3</td>
<td>Draw lines</td>
</tr>
<tr>
<td>xtDrawPoint/3</td>
<td>Draw point</td>
</tr>
<tr>
<td>xtDrawPoints/3</td>
<td>Draw points</td>
</tr>
<tr>
<td>xtDrawQueryDumpFile/3</td>
<td>Query size of memory dump</td>
</tr>
<tr>
<td>xtDrawQueryPointer/3</td>
<td>Query position of pointer</td>
</tr>
<tr>
<td>xtDrawReadWindowDump/8</td>
<td>Read memory dump</td>
</tr>
<tr>
<td>xtDrawRecallGC/2</td>
<td>Set graphics context</td>
</tr>
<tr>
<td>xtDrawRectangle/5</td>
<td>Draw rectangle</td>
</tr>
<tr>
<td>xtDrawRectangles/2</td>
<td>Draw rectangles</td>
</tr>
<tr>
<td>xtDrawRepaint/1</td>
<td>Restore window contents</td>
</tr>
<tr>
<td>xtDrawSegments/2</td>
<td>Draw line segments</td>
</tr>
<tr>
<td>xtDrawSetDashes/3</td>
<td>Set line pattern</td>
</tr>
<tr>
<td>xtDrawSetParam/3</td>
<td>Set parameters for graphics context</td>
</tr>
<tr>
<td>xtDrawSetRubberGC/1</td>
<td>Set rubber-banding mode</td>
</tr>
<tr>
<td>xtDrawSetTSOrigin/3</td>
<td>Set start point for pattern</td>
</tr>
<tr>
<td>xtDrawStoreGC/2</td>
<td>Save graphics context</td>
</tr>
<tr>
<td>xtDrawString/4</td>
<td>Output character string</td>
</tr>
<tr>
<td>xtDrawStringExtents/5</td>
<td>Query size of character string</td>
</tr>
<tr>
<td>xtDrawUnsetRubberGC/1</td>
<td>Cancel rubber-banding mode</td>
</tr>
<tr>
<td>xtDrawWriteWindowDump/2</td>
<td>Create memory dump of window contents</td>
</tr>
<tr>
<td>xtDrawXmString/8</td>
<td>Output compound string</td>
</tr>
<tr>
<td>xtDrawXmStringImage/8</td>
<td>Output compound string with foreground and background</td>
</tr>
<tr>
<td>xtDrawXmStringUnderline/9</td>
<td>Output underlined compound string</td>
</tr>
<tr>
<td>xtGetActionKeysym/2</td>
<td>Output underlined compound string</td>
</tr>
</tbody>
</table>
## Event loop

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmUpdateDisplay/1</td>
<td>Initiate execution of expose events</td>
</tr>
<tr>
<td>xtBreakMainLoop/0</td>
<td>Interrupt main loop</td>
</tr>
<tr>
<td>xtCallAcceptFocus/1</td>
<td>Interrupt main loop</td>
</tr>
<tr>
<td>xtCallActionProc/2</td>
<td>Interrupt main loop</td>
</tr>
<tr>
<td>xtLoopWhileEvents/0</td>
<td>Process outstanding events</td>
</tr>
<tr>
<td>xtMainLoop/0</td>
<td>Execute main event loop</td>
</tr>
<tr>
<td>xtPending/0</td>
<td>Check for pending events</td>
</tr>
<tr>
<td>xtProcessEvent/1</td>
<td>Process event</td>
</tr>
</tbody>
</table>

## FileSelectionBox

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmFileSelectionBoxGetChild/3</td>
<td>Retrieve components of the FileSelectionBox widget</td>
</tr>
<tr>
<td>xmFileSelectionDoSearch/2</td>
<td>Search file system for directory or file</td>
</tr>
</tbody>
</table>

## Flush output buffer

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtFlush/1</td>
<td>Flush X server output buffer</td>
</tr>
<tr>
<td>xtSync/1</td>
<td>Flush output buffer and execute all pending jobs</td>
</tr>
</tbody>
</table>

## Font List

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmFontListAppend/3</td>
<td>Append entries to a Font List</td>
</tr>
<tr>
<td>xmFontListCopy/2</td>
<td>Copy a Font List</td>
</tr>
<tr>
<td>xmFontListCreate/2</td>
<td>Create a new FontList from entry or list of entries</td>
</tr>
<tr>
<td>xmFontListFree/1</td>
<td>Free memory for a Font List</td>
</tr>
</tbody>
</table>

## Interface to the Window Manager

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmActivateProtocol/3</td>
<td>Activate protocol</td>
</tr>
<tr>
<td>xmActivateWMProtocol/2</td>
<td>Activate protocol</td>
</tr>
<tr>
<td>xmAddProtocolCallback/5</td>
<td>Define callback predicate for a protocol</td>
</tr>
<tr>
<td>xmAddProtocols/3</td>
<td>Define protocols</td>
</tr>
<tr>
<td>xmAddWMProtocolCallback/4</td>
<td>Define callback predicate for a WM protocol</td>
</tr>
</tbody>
</table>
Overview

- **xmAddWMProtocols**/2: Define WM protocols
- **xmDeactivateProtocol**/3: Deactivate protocol
- **xmDeactivateWMProtocol**/2: Deactivate protocol
- **xmGetAtomName**/2: Query the name of an atom registered with the X server
- **xmInternAtom**/3: Register atom with the X server or check if it exists
- **xmIsMotifWMRunning**/1: Check status of the Motif Window Manager
- **xmRemoveProtocolCallback**/5: Remove callback predicate from a protocol
- **xmRemoveProtocols**/3: Remove protocols
- **xmRemoveWMProtocolCallback**/4: Remove callback predicate from a WM protocol
- **xmRemoveWMProtocols**/2: Remove WM protocols
- **xmSendClientMessage**/5: Send message to window owner

Interrogation predicates

- # xtClass**/2: Query class and name of a widget
- # xtIsSubclass**/2: Check class to which a widget belongs
- # xtName**/2: Query name of widget
- # xtNameToWidget**/3: Convert widget name to window name
- # xtParent**/2: Find related parent and child widgets
- # xtWidget**/1: Query widget name
- # xtWindowToWidget**/2: Query window id or widget name
- **xmGetFocusWidget**/2: Determine focus widget within hierarchy of widget
- **xmGetTabGroup**/2: Query tab group
- **xmGetTabGroup**/2: Return tab group widget of widget
- **xmGetTearOffControl**/2: Return tear off widget of a menu
- **xmGetVisibility**/2: Determine visibility status of a widget with respect to ancestors
- **xmGetXmDisplay**/1: Return display widget of display
- **xmGetXmScreen**/1: Return screen widget of screen
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmIsTraversable/1</td>
<td>Identify whether widget can be traversed</td>
</tr>
<tr>
<td>xmListGetKbdItemPos/2</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmListPosSelected/2</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmListPosToBounds/6</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmListSetKbdItemPos/2</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmListUpdateSelectedList/1</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmListYToPos/3</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmTrackingEvent/4</td>
<td>Select a widget</td>
</tr>
<tr>
<td>xmTrackingLocate/4</td>
<td>Select a widget</td>
</tr>
<tr>
<td>xmVersion/2</td>
<td>Query OSF/Motif version</td>
</tr>
<tr>
<td>xmWidgetGetBaselines/2</td>
<td>Retrieve Baseline information for a widget</td>
</tr>
<tr>
<td>xmWidgetGetDisplayRect/2</td>
<td>Retrieve Display Rectangle information for a widget</td>
</tr>
<tr>
<td>xtDisplayHeight/2</td>
<td>Query height of a display in pixels</td>
</tr>
<tr>
<td>xtDisplayHeightMM/2</td>
<td>Query height of a display in pixels</td>
</tr>
<tr>
<td>xtDisplayWidthMM/2</td>
<td>Query width of a display in pixels</td>
</tr>
<tr>
<td>xtDisplayWidth/2</td>
<td>Query width of a display in pixels</td>
</tr>
<tr>
<td>xtHasCallbacks/2</td>
<td>Check existence of callbacks</td>
</tr>
<tr>
<td>xtIsComposite/1</td>
<td>Check that a widget belongs to the CompositeWidgetClass</td>
</tr>
<tr>
<td>xtIsManaged/1</td>
<td>Check management of a widget</td>
</tr>
<tr>
<td>xtIsRealized/1</td>
<td>Check that a widget is realized</td>
</tr>
<tr>
<td>xtIsSensitive/1</td>
<td>Check that a widget can be activated</td>
</tr>
<tr>
<td>xtName/2</td>
<td>Query toolkit name of widget</td>
</tr>
<tr>
<td>xtOverrideTranslations/2</td>
<td>Convert widget name to window name</td>
</tr>
<tr>
<td>xtTranslateCoords/5</td>
<td>Convert coordinates</td>
</tr>
<tr>
<td>xtUngrabButton/3</td>
<td>Convert coordinates</td>
</tr>
<tr>
<td>xtVersion/2</td>
<td>Query version of X Window System</td>
</tr>
<tr>
<td>xtWidgetList/1</td>
<td>Query widget names</td>
</tr>
</tbody>
</table>

### Keys and keyboard

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmAddTabGroup/1</td>
<td>Add widget to tab group</td>
</tr>
</tbody>
</table>
### Overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmProcessTraversal/2</code></td>
<td>Specify processing mode for keyboard events</td>
</tr>
<tr>
<td><code>xmRemoveTabGroup/1</code></td>
<td>Remove widget from tab group</td>
</tr>
<tr>
<td><code>xtAddGrab/3</code></td>
<td>Restrict input to a particular widget</td>
</tr>
<tr>
<td><code>xtGrabKey/3</code></td>
<td>Reserve keyboard and redirect keyboard event</td>
</tr>
<tr>
<td><code>xtGrabKey/7</code></td>
<td>Reserve keyboard and redirect keyboard event</td>
</tr>
<tr>
<td><code>xtGrabKeyboard/4</code></td>
<td>Reserve keyboard</td>
</tr>
<tr>
<td><code>xtGrabPointer/7</code></td>
<td>Reserve keyboard</td>
</tr>
<tr>
<td><code>xtInstallAccelerators/2</code></td>
<td>Specify accelerators to operate across windows</td>
</tr>
<tr>
<td><code>xtInstallAllAccelerators/2</code></td>
<td>Specify accelerators to operate across windows</td>
</tr>
<tr>
<td><code>xtKeysymToKeycode/3</code></td>
<td>Convert key code (Keycode) to key symbol (Keysym)</td>
</tr>
<tr>
<td><code>xtKeysymToKeycodeList/4</code></td>
<td>Convert key symbol (Keysym) to key code (Keycode) and vice versa</td>
</tr>
<tr>
<td><code>xtRemoveGrab/1</code></td>
<td>Remove input restrictions</td>
</tr>
<tr>
<td><code>xtSetKeyboardFocus/2</code></td>
<td>Redirect input events</td>
</tr>
<tr>
<td><code>xtUngrabKey/3</code></td>
<td>Cancel keyboard reservation</td>
</tr>
<tr>
<td><code>xtUngrabKeyboard/1</code></td>
<td>Cancel keyboard reservation</td>
</tr>
<tr>
<td><code>xtUngrabPointer/1</code></td>
<td>Cancel keyboard reservation</td>
</tr>
</tbody>
</table>

### List

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>XmListSetBottomPos/2</code></td>
<td>Specify last visible entry in a list</td>
</tr>
<tr>
<td><code>xmlListAddItem/3</code></td>
<td>Insert entry in a list</td>
</tr>
<tr>
<td><code>xmlListAddItemUnselected/3</code></td>
<td>Insert entry in a list</td>
</tr>
<tr>
<td><code>xmlListAddItems/3</code></td>
<td>Insert entries in a list</td>
</tr>
<tr>
<td><code>xmlListAddItemsUnselected/3</code></td>
<td>Insert entries in a list</td>
</tr>
<tr>
<td><code>xmlListDeleteAllItems/1</code></td>
<td>Delete all entries from a list</td>
</tr>
<tr>
<td><code>xmlListDeleteItem/2</code></td>
<td>Delete entry from a list</td>
</tr>
<tr>
<td><code>xmlListDeleteItems/2</code></td>
<td>Delete entries from a list</td>
</tr>
<tr>
<td><code>xmlListDeleteItemsPos/3</code></td>
<td>Delete entries from a list starting at a given position</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>xmListDeletePos/2</td>
<td>Delete entry from a list</td>
</tr>
<tr>
<td>xmListDeletePositions/2</td>
<td>Delete entries from a list</td>
</tr>
<tr>
<td>xmListDeselectAllItems/1</td>
<td>Cancel markings on entries in a list</td>
</tr>
<tr>
<td>xmListDeselectItem/2</td>
<td>Cancel marking on entry in a list</td>
</tr>
<tr>
<td>xmListDeselectPos/2</td>
<td>Cancel marking on entry in a list</td>
</tr>
<tr>
<td>xmListGetMatchPos/3</td>
<td>Query position of entries in a list</td>
</tr>
<tr>
<td>xmListGetSelectedPos/2</td>
<td>Query position of marked entries in a list</td>
</tr>
<tr>
<td>xmListItemExists/2</td>
<td>Check existence of an entry in a list</td>
</tr>
<tr>
<td>xmListItemPos/3</td>
<td>Query position of an entry in a list</td>
</tr>
<tr>
<td>xmListReplaceItems/3</td>
<td>Replace entries in a list</td>
</tr>
<tr>
<td>xmListReplaceItemsPos/3</td>
<td>Replace entries in a list</td>
</tr>
<tr>
<td>xmListReplaceItemsPosUnselected/3</td>
<td>Replace entries in a list without selection</td>
</tr>
<tr>
<td>xmListReplaceItemsUnselected/3</td>
<td>Replace entries in a list without selection</td>
</tr>
<tr>
<td>xmListReplacePositions/3</td>
<td>Replace entries in a list</td>
</tr>
<tr>
<td>xmListSelectItem/3</td>
<td>Mark entry in a list</td>
</tr>
<tr>
<td>xmListSelectPos/3</td>
<td>Mark entry in a list</td>
</tr>
<tr>
<td>xmListSetAddMode/2</td>
<td>Set add mode for inserting entries into a list</td>
</tr>
<tr>
<td>xmListSetBottomItem/2</td>
<td>Specify last visible entry in a list</td>
</tr>
<tr>
<td>xmListSetHorizPos/2</td>
<td>Specify position of the horizontal scroll bar</td>
</tr>
<tr>
<td>xmListSetItem/2</td>
<td>Specify first visible entry in a list</td>
</tr>
<tr>
<td>xmListSetPos/2</td>
<td>Specify first visible entry in a list</td>
</tr>
</tbody>
</table>

**MRM predicates**

- mrmCloseHierarchy/1: Close Motif Resource Manager hierarchy
- mrmFetchColorLiteral/4: Fetch color value from a UID hierarchy
- mrmFetchIconLiteral/6: Convert icon value from a UID hierarchy into a pixmap
- mrmFetchLiteral/4: Fetch value of a UIL variable
- mrmFetchSetValues/3: Set resource values equal
- mrmFetchWidget/4: Create UIL widget
### Overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mrnFetchWidgetOverride/5</td>
<td>Create UIL widget</td>
</tr>
<tr>
<td>mrnInitialize/0</td>
<td>Initialize Motif Resource Manager (MRM)</td>
</tr>
<tr>
<td>mrnOpenHierarchy/2</td>
<td>Create MRM hierarchy id and open associated files</td>
</tr>
<tr>
<td>mrnRegisterIdentifierNames/1</td>
<td>Generate assignments of UIL names and values</td>
</tr>
<tr>
<td>mrnRegisterIdentifierNamesInHierarchy/2</td>
<td>Generate assignments of UIL names and values</td>
</tr>
<tr>
<td>mrnRegisterProcedureNames/1</td>
<td>Generate assignments of UIL names to callback predicates</td>
</tr>
</tbody>
</table>

#### MainWindow

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmMainWindowSep1/2</td>
<td>Specify widget names of separators</td>
</tr>
<tr>
<td>xmMainWindowSep2/2</td>
<td>Specify widget names of separators</td>
</tr>
<tr>
<td>xmMainWindowSep3/2</td>
<td>Specify widget names of separators</td>
</tr>
<tr>
<td>xmMainWindowSetAreas/6</td>
<td>Specify children of the MainWindow widget</td>
</tr>
</tbody>
</table>

#### MessageBox

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmMessageBoxGetChild/3</td>
<td>Query a component of the MessageBox widget</td>
</tr>
</tbody>
</table>

#### Other event sources

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtAddInput/5</td>
<td>Set input predicate</td>
</tr>
<tr>
<td>xtAddTimeOut/4</td>
<td>Set TimeOut predicates</td>
</tr>
<tr>
<td>xtAddWorkProc/3</td>
<td>Set work predicate</td>
</tr>
<tr>
<td>xtAugmentTranslations/2</td>
<td>Set work predicate</td>
</tr>
<tr>
<td>xtRemoveInput/1</td>
<td>Cancel input job</td>
</tr>
<tr>
<td>xtRemoveTimeOut/1</td>
<td>Cancel timeout job</td>
</tr>
<tr>
<td>xtRemoveWorkProc/1</td>
<td>Cancel work job</td>
</tr>
</tbody>
</table>

#### Popup

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtCreatePopupShell/5</td>
<td>Create pop-up shell</td>
</tr>
<tr>
<td>xtPopdown/1</td>
<td>Terminate output of a pop-up widget</td>
</tr>
</tbody>
</table>
Overview

xtPopup/2
Output pop-up widget

xtPopupSpringLoaded/1
Output pop-up widget

**Representation Types**

xmRepTypeAddReverse/1
Install reverse converter

xmRepTypeGetNameList/3
Generate list of values for a representation type

xmRepTypeGetRecord/2
Generate list of data for a representation type

xmRepTypeGetRegistered/1
Generate list of representation type names

xmRepTypeInstallTearOffModelConverter/0
Install converter for RowColumn’s TearOffModel Resource

xmRepTypeRegister/3
Register a representation type resource

xmRepTypeValidValue/3
Test validity of numerical value for representation type resource

**Resources**

xtGetApplicationNameAndClass/3
Retrieve values of resources

xtGetApplicationResources/3
Retrieve values of resources

xtGetDefault/3
Retrieve values of resources

xtGetDefaultColorInfo/2
Retrieve values of resources

xtGetValue/2
Retrieve value of a resource

xtGetValues/2
Retrieve values of a resource

xtGetWidgetMapState/2
Retrieve value of a resource

xtGrabButton/9
Retrieve value of a resource

xtSetValue/2
Set and modify resource

xtSetValues/2
Set and modify resources

xtSetWidgetCursor/2
Set and modify resource

**RowColumn**

xmGetMenuCursor/1
Query current menu cursor

xmGetPostedFromWidget/2
Query menu environment

xmMenuPosition/1
Position pop-up menu

xmOptionButtonGadget/2
Get information about menu item
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmOptionLabelGadget/2</code></td>
<td>Get information about menu item</td>
</tr>
<tr>
<td><code>xmSetMenuCursor/1</code></td>
<td>Set menu cursor</td>
</tr>
<tr>
<td><code>xmScaleGetValue/2</code></td>
<td>Query position of slider along a scale</td>
</tr>
<tr>
<td><code>xmScaleSetValue/2</code></td>
<td>Set position of slider along a scale</td>
</tr>
<tr>
<td><code>xmScrollVisible/4</code></td>
<td>Make obscured widget or gadget visible</td>
</tr>
<tr>
<td><code>xmScrollBarGetValues/5</code></td>
<td>Query position of slider on scroll bar</td>
</tr>
<tr>
<td><code>xmScrollBarSetValues/5</code></td>
<td>Set position of slider on scroll bar</td>
</tr>
<tr>
<td><code>xmScrolledWindowSetAreas/4</code></td>
<td>Add or remove components of the ScrolledWindow widget</td>
</tr>
<tr>
<td><code>xmSelectionBoxGetChild/3</code></td>
<td>Query component of the SelectionBox widget</td>
</tr>
<tr>
<td><code>xmTextClearSelection/1</code></td>
<td>Cancel first marking in the Text widget</td>
</tr>
<tr>
<td><code>xmTextCopy/1</code></td>
<td>Copy text into Text widget clipboard</td>
</tr>
<tr>
<td><code>xmTextCut/1</code></td>
<td>Cut text into Text widget clipboard</td>
</tr>
<tr>
<td><code>xmTextDisableRedisplay/1</code></td>
<td>Prevent visual update of Text widget</td>
</tr>
<tr>
<td><code>xmTextEnableRedisplay/1</code></td>
<td>Force visual update of Text widget</td>
</tr>
<tr>
<td><code>xmTextFieldGetSubstring/4</code></td>
<td>Retrieve a portion of the text in a text field widget</td>
</tr>
<tr>
<td><code>xmTextFreeConstraints/1</code></td>
<td>Cancel restrictions on user inputs into the Text widget</td>
</tr>
</tbody>
</table>
Overview

xmTextGetBaseline/2  
Query position of first baseline in a Text widget

xmTextGetConstraints/5  
Query restrictions on user inputs into the Text widget

xmTextGetEditable/1  
Check that a text in the Text widget is editable

xmTextGetInsertionPosition/2  
Query position of cursor in the Text widget

xmTextGetLastPosition/2  
Query position of last character in the Text widget

xmTextGetMaxLength/2  
Query maximum length of text in the Text widget

xmTextGetSelection/2  
Query marked text in the Text widget

xmTextGetSelectionPosition/3  
Query position of marked text in the Text widget

xmTextGetString/2  
Query text in the Text widget

xmTextGetString/3  
Query text in the Text widget

xmTextGetString/4  
Query text in the Text widget

xmTextGetSubstring/4  
Retrieve a portion of the text in a text widget

xmTextGetTopCharacter/2  
Query position of first character in the Text widget

xmTextInsert/3  
Insert text into the Text widget

xmTextPaste/1  
Insert text from Text widget clipboard

xmTextPosToXY/4  
Query position of a character in the Text widget

xmTextRemove/1  
Delete text from the Text widget

xmTextReplace/4  
Replace text in the Text widget

xmTextScroll/2  
Move text in the Text widget

xmTextSetAddMode/2  
Set add mode for inserting text into Text widget

xmTextSetConstraints/5  
Specify restrictions on user inputs into Text widget

xmTextSetEditable/2  
Specify a text in the Text widget as editable

xmTextSetHighlight/4  
Set text highlighting in the Text widget
## Overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmTextSetInsertionPosition</code></td>
<td>Set position of cursor in the text widget</td>
</tr>
<tr>
<td><code>xmTextSetMaxLength</code></td>
<td>Set maximum length of character string in the Text widget</td>
</tr>
<tr>
<td><code>xmTextSetSelection</code></td>
<td>Set text marking in the Text widget</td>
</tr>
<tr>
<td><code>xmTextSetSource</code></td>
<td>Specify common sources for Text widgets</td>
</tr>
<tr>
<td><code>xmTextSetString</code></td>
<td>Specify contents of a Text widget</td>
</tr>
<tr>
<td><code>xmTextSetTopCharacter</code></td>
<td>Set position of first character in the Text widget</td>
</tr>
<tr>
<td><code>xmTextShowPosition</code></td>
<td>Make text visible in the Text widget</td>
</tr>
<tr>
<td><code>xmTextXYToPos</code></td>
<td>Query position of a character in the Text widget</td>
</tr>
</tbody>
</table>

## ToggleButton

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmToggleButtonGadgetGetState</code></td>
<td>Check status of ToggleButton gadget</td>
</tr>
<tr>
<td><code>xmToggleButtonGadgetSetState</code></td>
<td>Set status of ToggleButton gadget</td>
</tr>
</tbody>
</table>

## X environment initialization

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xtAddEditresSupport</code></td>
<td>Add support for editres client</td>
</tr>
<tr>
<td><code>xtInitialize</code></td>
<td>Initialize Toolkit Intrinsics</td>
</tr>
<tr>
<td><code>xtReInitialize</code></td>
<td>Release widgets</td>
</tr>
<tr>
<td><code>xtSetLanguageProc</code></td>
<td>Add internationalisation support</td>
</tr>
<tr>
<td><code>xtTopLevelShellWidget</code></td>
<td>Query widget name of toplevel widget</td>
</tr>
</tbody>
</table>
Chapter 6

Reference section on predefined predicates

In this chapter, all the predicates of the IF/Prolog interface for OSF/Motif are described in alphabetical sequence. Predicates which are alphabetically consecutive and have a similar function are described together.

The description of each predicate subdivides broadly into:

- Descriptive title
- Predicate head (boxed)
- Full description

The contents of these individual components are indicated in more detail below.

**Descriptive title**

The function of the predicate is described here in keywords.

**Predicate head (boxed)**

The predicate head, e.g.

```prolog
# xtNameToWidget( ?RefWidgetsName, +ToolkitName, ?WidgetName)
```

provides the most important information about the predicate:

- **Functor**
- The functor (in this example: `xtNameToWidget`) is highlighted in boldface.
Backtracking

Predicates which permit backtracking are indicated by a hash character '#' in front of the functor (as in the above example). This character is omitted for functors which do not support backtracking.

Arguments

The arguments (in the example: RefWidgetName, ToolkitName and WidgetName) are shown in normal type.

Call types

Each argument is preceded by one of the characters '+', '-', '?' or @, to indicate its call type. The meaning of the call types is described in the IF/Prolog User’s Guide [2], chapter entitled Database and procedure execution, section on Predicates.

Full description

Additional detailed information about the predicate will be found here. The following format conventions are used:

- Italics: Arguments, e.g. RefWidgetName.
- Normal type: Everything else; in particular, functors, structures and files are not highlighted in the description.

All the predicates are described in a uniform format. The full description begins with a passage of text which gives a detailed explanation of how the predicate works, and the function of its arguments. If two or more similar predicates are described together, their differences will be pointed out here. The various subsections have the headings:

- Arguments
- Exceptions
- Hints
- Caution
- Example
- See also

Some of these subsections may be omitted if they are inapplicable for the predicate concerned; for example, the subsection headed "Caution" appears in only a few cases.
Arguments

This shows for each argument the types of terms that are either required for instantiation (call types + and @) or permissible (call type ?) when the predicate is called. If only specific atoms are permitted as arguments, these are shown in the form of a list, separated by '|' symbols.

Exceptions

Possible runtime exceptions are described here. The exception messages are shown exactly as they are output by IF/Prolog. Type and mode exceptions are not listed here; they can be found in the appendix.

Caution

The information shown here should always be noted, as failure to comply may produce unintended results, some of which may be serious.

Hints

Explanatory details on the use and special features of the predicate are noted here.

Example

This subsection illustrates how to use the predicate.

See also

This subsection lists cross-references to other IF/Prolog predicates and to OSF/Motif and X-Windows functions.

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>Cross-reference to other IF/Prolog predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF/Motif</td>
<td>Cross-reference to OSF/Motif functions</td>
</tr>
<tr>
<td>X</td>
<td>Cross-reference to X functions</td>
</tr>
</tbody>
</table>
mrmCloseHierarchy/1 closes the hierarchy \textit{HierarchyId} of the Motif Resource Manager MRM. A hierarchy defines the sequence in which files will be searched by the MRM predicates. All the files assigned to the hierarchy will be closed and the storage space will be released.

\textit{HierarchyId} is the hierarchy id of an open hierarchy. This will have been defined previously using mrmOpenHierarchy/2.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>HierarchyId</td>
<td>Opaque</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \texttt{environment\_error(mrm\_function\_failed)}
  - It was not possible to successfully execute the MRM function concerned.
- \texttt{domain\_error(internal\_hierarchy\_id)}
  - An invalid value was specified for the HierarchyId argument.

**See also**

- IF/Prolog \texttt{mrmOpenHierarchy/2}
- OSF/Motif \texttt{MrmCloseHierarchy()}

Fetch color value from a UID hierarchy

\[
mrmFetchColorLiteral ( \ +\text{WidgetName}, +\text{HierarchyId}, +\text{UILName}, ?\text{Value} )
\]

\mrmFetchColorLiteral/4\ returns the color value \text{Value}\ of a UIL color with the name \text{UIL-Name}. This color value can be used if it is required to set a color value using \xtSetValues/2. \text{HierarchyId}\ is the hierarchy id of a hierarchy which is open. This will have been defined using \mrmOpenHierarchy/2.

The predicate will fail if \text{UILName}\ is not found in the UIL file.

Arguments

- \text{WidgetItem}\ Atom, the name of a widget
- \text{HierarchyId}\ Opaque
- \text{UILName}\ Atom, a UIL name
- \text{Value}\ Atom, a color

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \text{environment\_error(mrm\_function\_failed)}
  It was not possible to successfully execute the MRM function concerned.
- \text{domain\_error(internal\_hierarchy\_id)}
  An invalid value was specified for the HierarchyId argument.

Hints

The value returned is not the actual name of the color, but simply an internal name. It is valid only until the end of the IF/Prolog process.

See also

- IF/Prolog \ mrmOpenHierarchy/2
- OSF/Motif MrmFetchColorLiteral()
Convert icon value from a UID hierarchy into a pixmap

```
mrmFetchIconLiteral ( +WidgetName, +HierarchyId, +UILName, +Foreground, +Background, ?Value )
```

mrmFetchIconLiteral/6 converts an icon value of a UID hierarchy with the name $UILName$ into a pixmap for the widget called $WidgetName$.

mrmFetchIconLiteral/6 returns the pixmap value $Value$ using the foreground and background colors $Foreground$ and $Background$.

$HierarchyId$ is the hierarchy id for an open hierarchy. This will have been defined previously using mrmOpenHierarchy/2.

Arguments

- **WidgetItem** Atom, the name of a widget
- **HierarchyId** Opaque
- **UILName** Atom, the UIL name
- **Foreground** Atom, color identifier which is server-dependent (see RGB database).
- **Background** Atom, color identifier which is server-dependent (see RGB database).
- **Value** Atom, pixmap

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **environment_error(mrm_function_failed)**
  It was not possible to successfully execute the MRM function concerned.
- **domain_error(pixel_resource)**
  The value specified for the foreground or background is not permitted. Conversion to the X Toolkit representation will fail.
- **domain_error(internal_hierarchy_id)**
  An invalid value was specified for the HierarchyId argument.

Hints

The value returned is not a pixmap name in IF/Prolog format, but simply an internal name. It is valid only until the end of the IF/Prolog process.
See also

<table>
<thead>
<tr>
<th>System</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>mrmOpenHierarchy/2</td>
</tr>
<tr>
<td>OSF/Motif</td>
<td>MrmFetchIconLiteral()</td>
</tr>
</tbody>
</table>
Fetch value of a UIL variable

**mrmFetchLiteral** ( +WidgetName, +HierarchyId, +UILName, ?Value )

mrmFetchLiteral/4 returns the value Value of a UIL variable called UILName (VALUE declaration in UIL). This value is converted to its IF/Prolog format.

HierarchyId is the hierarchy id of an open hierarchy. This will have been defined previously using mrmOpenHierarchy/2.

The predicate will fail if UILName is not found in the UIL file.

UILName can have the following formats:

<table>
<thead>
<tr>
<th>UILName type</th>
<th>IF/Prolog type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPOUND_STRING_TABLE</td>
<td>List of XmStrings</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Atom, true or false</td>
</tr>
<tr>
<td>STRING</td>
<td>Atom</td>
</tr>
<tr>
<td>STRING_TABLE</td>
<td>List of atoms</td>
</tr>
<tr>
<td>INTEGER_TABLE</td>
<td>List of integers</td>
</tr>
<tr>
<td>COMPOUND_STRING</td>
<td>List, XmString</td>
</tr>
<tr>
<td>COMPOUND_STRING_TABLE</td>
<td>List of XmStrings</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Real number</td>
</tr>
<tr>
<td>FONT</td>
<td>Atom, a valid font</td>
</tr>
<tr>
<td>FONT_TABLE</td>
<td>Atom, a valid font list</td>
</tr>
<tr>
<td>TRANSLATION_TABLE</td>
<td>Atom, a valid translation table</td>
</tr>
</tbody>
</table>

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **HierarchyId**: Opaque
- **UILName**: Atom, a UIL name
- **Value**: Term, dependent on the UILName type

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**environment_error(mrm_function_failed)**

It was not possible to successfully execute the MRM function concerned.
domain_error(internal_hierarchy_id)
   An invalid value was specified for the HierarchyId argument.

representation_error(conversion_for_data_type)
   A UIL type has been used which cannot be converted to an
   IF/Prolog representation.

Hints

The value returned for font, font_table and translation_table is not in normal IF/Prolog
representation. The value returned is merely an internal identifier; this can be used in
later calls of xtSetValues/2. It is valid only until the end of the IF/Prolog process.

See also

IF/Prolog         mrmOpenHierarchy/2
OSF/Motif         MrmFetchLiteral()
Set resource values equal

\[ \text{mrmFetchSetValues} \left( +\text{HierarchyId}, +\text{WidgetName}, +\text{UILArgList} \right) \]

mrmFetchSetValues/3 matches the resource values of the widget called \text{WidgetName} to the values in \text{UILArgList}. This is a list which corresponds to the one passed to xtSetValues/2, except that it has UIL names in place of the resource values.

\text{HierarchyId} is the hierarchy id for an open widget hierarchy, as previously defined using mrmOpenHierarchy/2.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HierarchyId</td>
<td>Opaque</td>
</tr>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>UILArgList</td>
<td>List of value assignments for resources</td>
</tr>
<tr>
<td>ResourceName</td>
<td>Atom</td>
</tr>
<tr>
<td>UILName</td>
<td>Atom, the name of a UIL object</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **environment_error(mrm_function_failed)**
  It was not possible to successfully execute the MRM function concerned.

- **environment_error(mrm_partial_success)**
  It was possible to execute the predicate partially. (Motif 1.1 only)

- **domain_error(resource_value_pair)**
  The specified list elements must be value assignments.

- **domain_error(internal_hierarchy_id)**
  An invalid value was specified for the HierarchyId argument.

- **representation_error(too_many_arguments)**
  No more than 100 elements are permitted in UILArgList.

See also

- IF/Prolog       mrmOpenHierarchy/2
- OSF/Motif       MrmFetchSetValues()
Create and name a UIL widget

```prolog
mrmFetchWidget ( +HierarchyId, +UILName, +ParentName, ?WidgetItemName )
mrmFetchWidgetOverride ( +HierarchyId, +UILName, +ParentName, +ToolkitName, ?WidgetItemName )
```

$mrmFetchWidget/4$ creates a widget called $UILName$ from a UIL module, as a child of the widget called $ParentName$. This widget is automatically registered for use in IF/Prolog and is given a unique IF/Prolog widget name. On the other hand, the subordinate widgets will not be registered for use in IF/Prolog. IF/Prolog will be unaware of them until the first callback for them occurs.

$mrmFetchWidgetOverride/5$ creates a widget called $UILName$ from a UIL module, as a child of the widget called $ParentName$ and gives it the name $ToolkitName$. The widget thus supplied is automatically registered for use in IF/Prolog and is given a unique IF/Prolog widget name. On the other hand, the subordinate widgets will not be registered for use in IF/Prolog. IF/Prolog will be unaware of them until the first callback for them occurs.

$HierarchyId$ is the hierarchy id of an open widget hierarchy, as defined previously using $mrmOpenHierarchy/2$.

The predicate will fail if $UILName$ is not found in the UIL file.

**Arguments**

- **HierarchyId**: Opaque
- **UILName**: Atom, the UIL name of a widget
- **ParentName**: Atom, a widget name
- **ToolkitName**: Atom
- **WidgetItemName**: Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- `environment_error(mrm_function_failed)`
  
  It was not possible to successfully execute the MRM function concerned.

- `domain_error(internal_hierarchy_id)`
  
  An invalid value was specified for the $HierarchyId$ argument.
See also

- IF/Prolog  
  `mrmOpenHierarchy/2`
- OSF/Motif  
  `MrmFetchWidget()`, `MrmFetchWidgetOverride()`
mrmInitialize/0 initializes the Motif Resource Manager (MRM). This gives access to the User Interface Language (UIL).

**See also**

OSF/Motif MrmInitialize()
Create MRM hierarchy id and open associated files

```prolog
mrmOpenHierarchy ( +FileList, ?HierarchyId )
```

mrmOpenHierarchy/2 creates a file hierarchy and assigns the files in the list called `FileList` to the hierarchy. The files are opened. They should be translated UID files (UID = User Interface Definition). The files containUIL definitions.

The predicate returns the hierarchy id `HierarchyId`.

This predicate is used to define the hierarchy of the files which will be accessed by the Motif Resource Manager’s predicates.

`HierarchyId` is the hierarchy id of the hierarchy thus created. It is passed to all the predicates which access the UIL user interface.

**Arguments**

- **FileList** List of atoms
  These atoms are file names
- **HierarchyId** Opaque

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- `environment_error(mrm_function_failed)`
  It was not possible to successfully execute the MRM function concerned.
- `environment_error(file_not_found)`
  The specified file cannot be opened for reading.

**See also**

- OSF/Motif MrmOpenHierarchyId()
mrmRegisterIdentifierNames/1/2

Generate assignment of UIL names and values

**mrmRegisterIdentifierNames** ( +ValueList )

**mrmRegisterIdentifierNamesInHierarchy** ( +HierarchyId, +ValueList )

mrmRegisterIdentifierNames/1 generates the assignments of UIL names to actual values in the program.

mrmRegisterIdentifierNamesInHierarchy/2 generates the assignments of UIL names to values for the hierarchy called HierarchyId. This predicate differs from mrmRegisterIdentifierNames/1 in that the range of UIL names is restricted to the hierarchy specified by HierarchyId.

The value UILValue is assigned to the UIL variable UILName.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HierarchyId</td>
<td>Opaque</td>
</tr>
<tr>
<td>ValueList</td>
<td>A list of lists</td>
</tr>
<tr>
<td>UILName</td>
<td>Atom</td>
</tr>
<tr>
<td>UILValue</td>
<td>Atom or integer</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **environment_error(mrm_function_failed)**
  
  It was not possible to successfully execute the MRM function concerned.

- **domain_error(internal_hierarchy_id)**
  
  An invalid value was specified for the HierarchyId argument.

**See also**

OSF/Motif
MrmRegisterNames(), ...NamesInHierarchy()
Generate assignment of UIL names to callback predicates

**mrmRegisterProcedureNames** ( +CallbackPredicateList )

mrmRegisterProcedureNames/1 generates assignments of UIL-conforming callback names to callback predicates.

CallbackPredicate/3 is called by converting the UIL callback tag (client data for the callback) so that it corresponds to the atom TagType. At present, the Motif Resource Manager does not support all UIL types.

The following are permissible values for TagType:

<table>
<thead>
<tr>
<th>TagType</th>
<th>Resulting IF/Prolog type for client data</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>float</td>
<td>real number</td>
</tr>
<tr>
<td>boolean</td>
<td>true or false</td>
</tr>
<tr>
<td>string</td>
<td>atom</td>
</tr>
<tr>
<td>ascii_table</td>
<td>list of atoms</td>
</tr>
<tr>
<td>color</td>
<td>atom, color</td>
</tr>
<tr>
<td>compound_string</td>
<td>list, the components of which are</td>
</tr>
<tr>
<td></td>
<td>compound strings</td>
</tr>
<tr>
<td>string_table</td>
<td>list of compound strings</td>
</tr>
<tr>
<td>font</td>
<td>atom, a valid font</td>
</tr>
<tr>
<td>font_table</td>
<td>atom, a valid font table</td>
</tr>
<tr>
<td>translation_table</td>
<td>atom, a valid translation table</td>
</tr>
<tr>
<td>integer_table</td>
<td>list containing integers</td>
</tr>
<tr>
<td>widget</td>
<td>atom, the name of a widget</td>
</tr>
<tr>
<td>icon</td>
<td>atom, a pixmap</td>
</tr>
</tbody>
</table>

**Arguments**

CallbackPredicateList Alist of lists The latter consist of the following three elements:

[UILName, CallbackPredicate, TagType]

UILName Atom
CallbackPredicate Atom, the functor of a callback predicate of arity three
TagType Atom

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).
mrmRegisterProcedureNames/1

Predicates

environment_error(mrm_function_failed)
It was not possible to successfully execute the MRM function concerned.

domain_error(tag_type)
The specified TagType is not permitted.

environment_error(too_many_procedures_registered)
A maximum of 30 MRM callback procedures is permitted.

Hints

The value returned for font, font_table and translation_table is not in normal IF/Prolog representation. The value returned is merely an internal identifier; this can be used in later calls of xtSetValues/2. It is invalid after the IF/Prolog process has ended.

See also

IF/Prolog  mrmFetchLiteral/4
OSF/Motif  MrmRegisterNames()
Activate protocols

xmActivateProtocol ( +ShellName, +Property, +Protocol )

xmActivateWMProtocol ( +ShellName, +Protocol )

These predicates activate protocols for inter-client communications.

xmActivateProtocol/3 activates the protocol called Protocol and re-enters it in the protocol list which is linked under the name Property to the Shell widget called ShellName.

xmActivateWMProtocol/2 activates the protocol called Protocol and re-enters it in the pre-defined protocol list, WM-PROTOCOLS, of the Shell widget called ShellName. The call is equivalent to:

xmActivateProtocol( ShellName, 'WM_PROTOCOLS', Protocol).

Information about the interaction between clients and the Window Manager will be found in [20] on ICCCM Inter-Client Communication Conventions.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShellName</td>
<td>Atom, the name of a shell widget</td>
</tr>
<tr>
<td>Property</td>
<td>Atom</td>
</tr>
<tr>
<td>Protocol</td>
<td>Atom</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog

xmDeactivateProtocol/3, xmDeactivateWMProtocol/2,

xmAddProtocols/3, xmSendClientMessage/5

OSF/Motif

XmActivateProtocol(), ...WMProtocol()
Define callback predicate for a protocol

\texttt{xmAddProtocolCallback ( +ShellName, +Property, +Protocol,}
\texttt{ +CallbackPredicate, @ClientData )}

This predicate defines callback predicates for a protocol.

\texttt{xmAddProtocolCallback/5} adds the callback predicate \textit{CallbackPredicate} to the internal callback list of the protocol identified by \textit{Protocol} and \textit{Property}, for the \textit{ShellName} widget. If the protocol is not yet registered, this is done automatically by means of the predicate \texttt{xmAddProtocols/3}.

The callback predicate \textit{CallbackPredicate} will be activated when a client message is received for this protocol.

The argument \textit{ClientData} is passed to the callback predicate when the call occurs.

**Arguments**

- \texttt{ShellName} Atom, the name of a shell widget
- \texttt{Property} Atom
- \texttt{Protocol} Atom
- \texttt{CallbackPredicate} Atom, the functor of a callback predicate of arity three
- \texttt{ClientData} Term

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \texttt{xmAddProtocols/3, xmSendClientMessage/5}
- OSF/Motif \texttt{XmAddProtocolCallback()}

---

IF/Prolog V5.2 209 Motif Manual
Define protocols

```
xmAddProtocols( +ShellName, +Property, +ProtocolList )
```

This predicate registers protocols for inter-client communications.

xmAddProtocols/3 passes the protocols in `ProtocolList` to the Protocol Manager and registers them in the protocol list which is linked to the Shell widget `ShellName` under the name `Property`.

Arguments

- `ShellName` Atom, the name of a shell widget
- `Property` Atom
- `ProtocolList` A list of atoms

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog `xmAddProtocolCallback/5`
- OSF/Motif `XmAddProtocols()`
Specify function for converting character strings into valid units of measurement

xmAddStringToUnitTypeConverter

xmAddStringToUnitTypeConverter/0 registers the OSF/Motif function XmCvtStringToUnitType(), which converts characters strings into units of measurement which are used for the geometry data of widgets. Examples of units of measurement are pixels, hundredths of a millimeter, thousandths of an inch, etc.

The conversion function is required if the type of the units of measurement used for a widget is to be specified via a resource or from the command line. The conversion function must be registered before the widget is created.

The toplevel widget is used to define the display.

See also

IF/Prolog          xmConvertUnits/6
OSF/Motif          XmCvtStringToUnitType(), XtAddConverter()
Add a widget to a tab group

\texttt{xmAddTabGroup ( +TabGroupName )}

\texttt{xmAddTabGroup/1} adds the widget called \textit{TabGroupName} to the list of tab groups. If the keyboard is used to navigate through a hierarchy of widgets, it is possible to put together widgets from the Manager or Primitive class of widget to form tab groups. Within any tab group, the arrow keys can be used to move the input focus on to the next widget in the group.

\textbf{Arguments}

\begin{itemize}
  \item \texttt{TabGroupName} \hspace{1cm} \texttt{Atom}, the name of a widget
\end{itemize}

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

\begin{itemize}
  \item IF/Prolog \hspace{1cm} \texttt{xmRemoveTabGroup/1}
  \item OSF/Motif \hspace{1cm} \texttt{XmAddTabGroup()}
\end{itemize}
Define callback predicate for a WM protocol

```
xmAddWMProtocolCallback ( +ShellName, +Protocol, +CallbackPredicate, @ClientData )
```

This predicate defines callback predicates for a protocol.

`xmAddWMProtocolCallback/4` adds the callback predicate `CallbackPredicate` to the internal callback list of the protocol called `Protocol` which is registered in the protocol list, WM_PROTOCOLS, for the `ShellName` widget. If the protocol is not registered it is automatically registered by `xmAddWMProtocols/3`.

The callback predicate `CallbackPredicate` will be activated when a client message is received for this protocol.

The argument `ClientData` is passed to the callback predicate when the call is made.

**Arguments**

- **ShellName**: Atom, the name of a shell widget
- **Protocol**: Atom
- **CallbackPredicate**: Atom, the functor of a callback predicate with an arity of three
- **ClientData**: Term

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmAddProtocolCallback/5, xmSendClientMessage/5`
- OSF/Motif: `XmAddWMProtocolCallback()`
Define WM protocols

\[ \text{xmAddWMProtocols} ( \text{+ShellName}, \text{+ProtocolList} ) \]

This predicate registers protocols for inter-client communications.

\( \text{xmAddWMProtocols/2} \) passes the protocols in \textit{ProtocolList} to the Protocol Manager and registers them in the predefined protocol list, \texttt{WM_PROTOCOLS}, of the Shell widget \texttt{Shell-Name}. The call is equivalent to:

\[
\text{xmAddProtocols( ShellName, 'WM_PROTOCOLS', ProtocolList ).}
\]

**Arguments**

- \texttt{ShellName} \hspace{1cm} \text{Atom, the name of a shell widget}
- \texttt{ProtocolList} \hspace{1cm} \text{A list of atoms}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \hspace{1cm} \texttt{xmlAddProtocols/3}
- OSF/Motif \hspace{1cm} \texttt{XmAddWMProtocols()}

Motif Manual 214 IF/Prolog V5.2
Draw highlight for the CascadeButton

**xmCascadeButtonGadgetHighlight** ( +WidgetItem, +Highlight )

**xmCascadeButtonHighlight** ( +WidgetItem, +Highlight )

These predicates create or delete a shadow highlight around a CascadeButton widget or gadget.

**Arguments**

- **WidgetItem**: Atom, the name of a widget
- **Highlight**: Atom: true | false | yes | no | on | off

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmlCascadeButtonHighlight()
Predicates xmClipboardCancelCopy/2

Cancel copying of data into the clipboard

xmClipboardCancelCopy ( +WidgetName, +ItemId )

xmClipboardCancelCopy/2 terminates a data copying operation which was started by xm-ClipboardStartCopy/3, without the clipboard being modified by any call of xmClipboardCopy/5 or xmClipboardCopyName/7 which has already taken place.

WidgetName may be any widget of the application, although it should be the same as the one in the associated call of xmClipboardStartCopy/3.

ItemId is the identifier returned by xmClipboardStartCopy/3.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ItemId</td>
<td>Opaque</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

environment_error(clipboard_locked)

The clipboard has been locked by another application.

Hints

Following the call of xmClipboardCancelCopy/2, ItemId becomes invalid and may no longer be used.

See also

<table>
<thead>
<tr>
<th>Environment</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xmClipboardStartCopy/3</td>
</tr>
<tr>
<td>OSF/Motif</td>
<td>XmClipboardCancelCopy()</td>
</tr>
</tbody>
</table>
Start copying data into the clipboard

xmClipboardCopy( +WidgetName, +ItemId, +FormatName, +Data, +PrivateId )

xmClipboardCopy/5 initiates the copying of data into the clipboard. However, the clipboard will not actually be modified until xmClipboardEndCopy/2 is called.

WidgetItem is any widget of the application, although it should be the same one as specified in the associated call of xmClipboardStartCopy/3.

ItemId is the identifier returned by xmClipboardStartCopy/3.

FormatName is a format specification which is stored with the data and is then used for retrieving (converting) the data. The format which is used must have been registered (see xmRegisterFormat/3). Two formats are predefined by IF/Prolog and are automatically registered:

- prolog_atom
  The data is specified as an atom, and is also returned by xmClipboardRetrieve/4 in this form.
- prolog_atom_list
  The data is specified as a list of atoms, and is also returned by xmClipboardRetrieve/4 in this form.

All other formats are currently handled by IF/Prolog as atoms.

Data specifies the data to be saved in the form defined in FormatName.

PrivateId is an arbitrary atom which is saved with the data in the clipboard.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ItemId</td>
<td>Opaque</td>
</tr>
<tr>
<td>FormatName</td>
<td>Atom</td>
</tr>
<tr>
<td>Data</td>
<td>Atom, or a list of atoms</td>
</tr>
<tr>
<td>PrivateId</td>
<td>Atom</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- environment_error(clipboard_fail)
  It was not possible to successfully execute the predicate.
- environment_error(clipboard_locked)
  The clipboard has been locked by another application.
Hints

Most of the data formats which are predefined in the X Window System cannot currently be used under IF/Prolog.

See also

IF/Prolog  
- `xmClipboardCopyByName/5`
- `xmClipboardEndCopy/2`
- `xmClipboardStartCopy/3`

OSF/Motif  
- `XmClipboardCopy()`
Copy data into the clipboard after a delay

```
xmClipboardCopyByName ( +WidgetName, +DataId, +FormatName, +Data, +PrivateId )
```

`xmClipboardCopyByName/5` carries out a data copying action which has been notified and delayed. This is performed by the callback predicate specified in `xmClipboardCopyName/7`.

`WidgetName` is any widget of the application, although it should be the same one as specified in the associated call of `xmClipboardStartCopy/3`.

`DataId` is the identifier used to notify a delayed data copying action. It is returned by `xmClipboardCopyByName/7`.

`FormatName` is a format specification which is stored with the data and is then used for retrieving (converting) the data. The format which is used must have been registered (see `xmRegisterFormat/3`). Two formats are predefined by IF/Prolog and are automatically registered:

- `prolog_atom` The data is specified as an atom, and is also returned by `xmClipboardRetrieve/4` in this form.
- `prolog_atom_list` The data is specified as a list of atoms, and is also returned by `xmClipboardRetrieve/4` in this form.

All other formats are currently handled by IF/Prolog as atoms.

`Data` specifies the data to be saved in the form defined in `FormatName`.

`PrivateId` is an arbitrary atom which is saved with the data in the clipboard.

**Arguments**

- `WidgetName` Atom, the name of a widget
- `DataId` Opaque
- `FormatName` Atom
- `Data` Atom or list of atoms
- `PrivateId` Atom

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- `environment_error(clipboard_locked)` The clipboard has been locked by another application.
**domain_error(clipboard_data)**

The specified argument contains non-convertible data (e.g. uninstantiated variables).

**type_error(clipboard_data_type)**

The Data argument must have the data type specified in the argument FormatName.

**See also**

- IF/Prolog       `xmClipboardCopy/5, xmClipboardCopyName/7`
- OSF/Motif       `xmClipboardCopyByName()`
Start delayed copying of data into the clipboard

```
xmClipboardCopyName ( +WidgetName, +ItemId, +FormatName,
                     +ClipboardPredicate, +Length, +PrivateId, -DataId )
```

xmClipboardCopyName/7 notifies delayed copying of data to the clipboard. This is used by the application to indicate that it is ready to transfer the data into the clipboard, if required (interrogation of the clipboard). This delayed copying of the data is then carried out by xmClipboardCopyByName/5. The notification of the delayed data copying action is not entered into the clipboard until xmClipboardEndCopy is called.

*WidgetItem* is any widget of the application, but should be the same one as specified in the associated call of xmClipboardStartCopy/3.

*ItemId* is the identifier returned by xmClipboardStartCopy/3.

*FormatName* is a format specification which is stored with the data, and is then used for retrieving (converting) the data. The format which is used must have been registered (see xmRegisterFormat/3). Two formats are predefined by IF/Prolog and are automatically registered:

- **prolog_atom**: The data is specified as an atom, and is also returned by xmClipboardRetrieve/4 in this form.
- **prolog_atom_list**: The data is specified as a list of atoms, and is also returned by xmClipboardRetrieve/4 in this form.

All other formats are currently handled by IF/Prolog as atoms.

*ClipboardPredicate* is a callback predicate with an arity of four, which is called when requests for data are received by the clipboard, or when the delayed data copying action is no longer required because the clipboard has been overwritten.

```
ClipboardPredicate( Widget, DataId, PrivateId, Reason )
```

The first three arguments match the corresponding parameters in the associated call of xmClipboardCopyName/7.

*Reason* specifies the reason for the call:

- **data_request**: The notified data has been requested from the clipboard. The callback predicate should copy the notified data using xmClipboardCopyByName/5.
- **delete**: The notified data is no longer required, because the clipboard has been overwritten.

If a variable is specified for *ClipboardPredicate*, then a predefined callback predicate for the interactive input of data will be called:
xmClipboardCopyByName(WidgetName, DataId, Format, Data, PrivateId).

xmClipboardCopyByName(WidgetName, DataId, Data, PrivateId).

Length specifies the length (in bytes) of the data of which the delayed copy is to be made, so that the clipboard can reserve appropriate space for it.

PrivateId is an arbitrary atom which is saved with the data in the clipboard.

DataId is an identifier which is returned to make it possible at a later point to refer to this notification (e.g. using xmClipboardWithdrawFormat/2). This identifier becomes invalid after ClipboardPredicate has been called, or after the call of xmClipboardWithdrawFormat/2, and may then no longer be used.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ItemId</td>
<td>Opaque</td>
</tr>
<tr>
<td>FormatName</td>
<td>Atom</td>
</tr>
<tr>
<td>ClipboardPredicate</td>
<td>Atom, the functor of a predicate with an arity of four, or a variable</td>
</tr>
<tr>
<td>Length</td>
<td>Integer</td>
</tr>
<tr>
<td>PrivateId</td>
<td>Atom</td>
</tr>
<tr>
<td>DataId</td>
<td>Opaque</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).
environment_error(clipboard_fail)
It was not possible to successfully execute the predicate.

environment_error(clipboard_locked)
The clipboard has been locked by another application.

See also
IF/Prolog           xmClipboardCopyByName/5
OSF/Motif           XmClipboardCopy()
End copying of data into the clipboard

```prolog
xmClipboardEndCopy ( +WidgetName, +ItemId )
```

`xmClipboardEndCopy/2` terminates a data copying action initiated by `xmClipboardStartCopy/3`. This has the effect that the clipboard is modified by the data or the notifications provided by any calls of `xmClipboardCopy/5` or `xmClipboardCopyName/7` which occurred between the start and end predicates. During this time the clipboard is locked against other applications.

`WidgetName` is any widget of the application, but should be the same one as specified in the associated call of `xmClipboardStartCopy/3`.

`ItemId` is the identifier returned by `xmClipboardStartCopy/3`.

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **ItemId**: Opaque

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- `environment_error(clipboard_fail)`
  - It was not possible to successfully execute the predicate.
- `environment_error(clipboard_locked)`
  - The clipboard has been locked by another application.

**See also**

- IF/Prolog: `xmClipboardCopy/5`, `xmClipboardStartCopy/3`
- OSF/Motif: `XmClipboardEndCopy()`
End copying of data out of the clipboard

```prolog
xmClipBoardEndRetrieve ( +WidgetName )
```

`xmClipBoardEndRetrieve/1` ends the copying of data from the clipboard. In doing this, it cancels the lock on the clipboard which was imposed by `xmClipBoardStartRetrieve/1`.

`WidgetName` is any widget of the application, but should be the same one as specified in the associated call of `xmClipBoardStartCopy/3`.

**Arguments**

`WidgetName` Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

```prolog
environment_error(clipboard_locked)
```

The clipboard has been locked by another application.

**Hints**

This predicate is not normally required in IF/Prolog, because, unlike in C, the complete data is already supplied by `xmClipBoardRetrieve/4`.

**See also**

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th><code>xmClipBoardRetrieve/4</code>, <code>xmClipBoardStartRetrieve/1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF/Motif</td>
<td>XmClipBoardEndRetrieve()</td>
</tr>
</tbody>
</table>
Query the number of formats in the clipboard

\[
xmClipBoardInquireCount \left( +\text{WidgetName}, \text{?Count} \right)
\]

\( \text{xmClipBoardInquireCount/2} \) returns \( \text{Count} \), the number of different data formats for which data is available in the clipboard. Their names can be obtained using \( \text{xmClipBoardInquireFormat/3} \). If there are no formats, and hence no data, the value 0 is returned.

\( \text{WidgetName} \) is any widget of the application, but should be the same one as specified in the associated call of \( \text{xmClipBoardStartCopy/3} \).

**Arguments**

- **WidgetName** Atom, the name of a widget
- **Count** Integer

**Exceptions**

- **environment_error(clipboard_no_data)**
  The clipboard contains no data elements which correspond to the requested format.

- **environment_error(clipboard_locked)**
  The clipboard has been locked by another application.

**See also**

- OSF/Motif XmClipBoardInquireCount()
Query format name for data in the clipboard

```
xmClipboardInquireFormat( +WidgetName, +Index, ?FormatName )
```

`xmClipboardInquireFormat/3` returns the names (`FormatName`) of formats for which data is available in the clipboard.

*WidgetName* is any widget of the application, but should be the same one as specified in the associated call of `xmClipboardStartCopy/3`.

*Index* is the number of an available format. This number must lie between 1 and the maximum number, as returned by `xmClipboardInquireCount/2`.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Index</td>
<td>Integer &gt; 0</td>
</tr>
<tr>
<td>FormatName</td>
<td>Atom</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **environment_error(clipboard_locked)**
  - The clipboard has been locked by another application.

- **environment_error(clipboard_no_data)**
  - The clipboard contains no data elements which correspond to the requested format.

**See also**

- IF/Prolog  
  - `xmClipboardInquireCount/2`
- OSF/Motif  
  - XmClipboardInquireFormat()
Query length of the data in the clipboard

```prolog
xmClipboardInquireLength ( +WidgetName, +FormatName, ?Length )
```

xmClipboardInquireLength/3 returns the length $Length$ of the stored data for the format $FormatName$.

$WidgetName$ is any widget of the application, but should be the same one as specified in the associated call of xmClipboardStartCopy/3.

**Arguments**

- **WidgetItem**: Atom, the name of a widget
- **FormatName**: Atom
- **Length**: Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **environment_error(clipboard_locked)**: The clipboard has been locked by another application.
- **environment_error(clipboard_no_data)**: The clipboard contains no data elements which correspond to the requested format.

**See also**

- IF/Prolog: xmClipboardInquireFormat/3
- OSF/Motif: XmClipboardInquireLength()
Query pending copy operations

```
xmClipboardInquirePendingItems( +WidgetName, +FormatName, ?ItemList )
```

xmClipboardInquirePendingItems/3 returns `ItemList`, information about notified and delayed data copying operations for the format `FormatName`.

`WidgetName` is any widget of the application, but should be the same one as specified in the associated call of `xmClipboardStartCopy/3`.

`FormatName` is a format specification which is stored with the data, and is then used for retrieving (converting) the data. The format used must be registered (see `xmRegisterFormat/3`). Two formats are predefined by IF/Prolog and are automatically registered:

- `prolog_atom` The data is specified as an atom, and is also returned by `xmClipboardRetrieve/4` in this form.
- `prolog_atom_list` The data is specified as a list of atoms, and is also returned by `xmClipboardRetrieve/4` in this form.

All other formats are currently handled by IF/Prolog as atoms.

`ItemList` contains a two-element sublist for each delayed data copying operation which has been notified. This sublist consists of the associated identifier `DataId` and the user data `PrivateId` which is linked to it.

**Arguments**

- **WidgetItem**: Atom, the name of a widget
- **FormatName**: Atom
- **ItemList**: List of lists
  - Each of these consists of two elements: [DataId, PrivateId].
- **DataId**: Opaque
- **PrivateId**: Atom

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

`environment_error(clipboard_locked)` The clipboard has been locked by another application.

**See also**

- IF/Prolog: `xmClipboardCopyName/7`
- OSF/Motif: `XmClipboardInquirePendingItems()`
Lock clipboard

```
xmClipboardLock( +WidgetName )
```

xmClipboardLock/1 locks the clipboard against access by other applications. This can be done repeatedly in an application.

xmClipboardUnlock/2 unlocks the clipboard.

`WidgetName` is any widget of the application, but should be the same one as specified in the associated call of xmClipboardStartCopy/3.

Arguments

- `WidgetName` : Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

```
environment_error(clipboard_locked)
```

The clipboard has been locked by another application.

Hints

This predicate is not normally required in IF/Prolog, because, unlike in C, the complete data is supplied by xmClipboardRetrieve/4.

See also

- IF/Prolog  `xmClipboardUnlock/2`
- OSF/Motif  `XmClipboardLock()`
Register the format name for the clipboard

```
xmClipboardRegisterFormat( +WidgetName, +FormatName, +FormatLength )
```

xmClipboardRegisterFormat/3 registers additional formats, for which data can then be
stored in the clipboard.

*WidgetName* is any widget of the application, but should be the same one as specified in the
associated call of xmClipboardStartCopy/3.

*FormatLength* specifies the size of the smallest unit (in bits) of which the data items specified
by *FormatName* can be made up. This allows the data to be exchanged between different
hardware platforms (e.g. with different byte arrangements). Only string-type formats can
usefully be used in IF/Prolog at the present time. For this reason, the value 8 should
normally be specified for *FormatLength*.

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **FormatName**: Atom
- **FormatLength**: Integer: 8 | 16 | 32

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **environment_error(illegal_format_length)**
  The only values which may be specified for the argument are 8,
  12 or 32.
- **domain_error(clipboard_bad_format)**
  The format could not be registered, e.g. because it already exists.
- **environment_error(clipboard_locked)**
  The clipboard has been locked by another application.

**See also**

- OSF/Motif XmClipboardRegisterFormat()
Retrieve data from the clipboard

xmClipboardRetrieve ( +WidgetName, +FormatName, ?Data, ?PrivateId )

xmClipboardRetrieve/4 retrieves data with the format called FormatName from the clipboard. All the data associated with the specified format will always be supplied.

WidgetName is any widget of the application, but should be the same one as specified in the associated call of xmClipboardStartCopy/3.

FormatName is a format specification which is stored with the data, and is then used for retrieving (converting) the data. The format must be registered (see xmRegisterFormat/3). Two formats are predefined by IF/Prolog and are automatically registered:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prolog_atom</td>
<td>The data is specified as an atom, and is also returned by xmClipboardRetrieve/4 in this form.</td>
</tr>
<tr>
<td>prolog_atom_list</td>
<td>The data is specified as a list of atoms, and is also returned by xmClipboardRetrieve/4 in this form.</td>
</tr>
</tbody>
</table>

All other formats are currently handled by IF/Prolog as atoms.

Data returns the data held in the clipboard for FormatName. For the predefined format prolog_atom_list, this is a list of atoms. For all other formats, one atom will be returned.

PrivateId returns the user data linked to Data.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>FormatName</td>
<td>Atom</td>
</tr>
<tr>
<td>Data</td>
<td>Atom or list of atoms</td>
</tr>
<tr>
<td>PrivateId</td>
<td>Atom</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

environment_error(clipboard_locked)

The clipboard has been locked by another application.

environment_error(clipboard_no_data)

The clipboard contains no data elements which correspond to the requested format.
See also

<table>
<thead>
<tr>
<th>platform</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xmClipboardEndRetrieve/1, xmClipboardStartRetrieve/1</td>
</tr>
<tr>
<td>OSF/Motif</td>
<td>XmClipboardRetrieve()</td>
</tr>
</tbody>
</table>
Prepare to copy data into the clipboard

xmClipboardStartCopy ( +WidgetName, +ClipLabel, -ItemId )

xmClipboardStartCopy/3 prepares for the copying of data into the clipboard. A call of this predicate is essential before the following predicates can be called:

xmClipboardCancelCopy/2, xmClipboardCopy/5, xmClipboardCopyName/7, xmClipboardEndCopy/2.

The identifier ItemId returned by the predicate must be passed to the above predicates. When xmClipboardEndCopy/2 or xmClipboardCancelCopy/2 is then called, the identifier ItemId becomes invalid and may not be used again. Between the calls of xmClipboardStartCopy/3 and xmClipboardEndCopy/2, calls of xmClipboardCopy/5 or xmClipboardCopyName/7 can be used to initiate the copying of data into the clipboard. However, the clipboard itself will not be modified until xmClipboardEndCopy/2 is called.

The data copying may take two forms.
xmClipboardCopy/5 initiates direct copying of the data. The data which is transferred is copied directly into the clipboard when xmClipboardEndCopy/2 is called.
xmClipboardCopyByName/5 initiates delayed copying of the data. When the associated call of xmClipboardEndCopy/2 is made, only a notification of the required copy is entered in the clipboard. Not until the data is retrieved from the clipboard (see xmClipboardRetrieve/4) is the actual data copy made by the associated callback. This copy can be executed directly in the callback predicate by means of xmClipboardByName/5.

WidgetName is any widget of the application, but should be the same one as specified in the associated call of xmClipboardStartCopy/3.

ClipLabel is a compound string which identifies the matching entry in the clipboard (e.g. the name of the application).

ClipLabel can be used by other X clients to assign the data to an application. It is not used within the application.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ClipLabel</td>
<td>Atom or a list, a valid XmString</td>
</tr>
<tr>
<td>ItemId</td>
<td>Opaque</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

environment_error(clipboard_locked)

The clipboard has been locked by another application.
See also

IF/Prolog  xmClipboardCopy/5, xmClipboardCopyName/7,
          xmClipboardEndCopy/2
OSF/Motif  XmClipboardCopy()
Predicates

xmClipboardStartRetrieve/1

Start to copy data from the clipboard

```
xmlClipboardStartRetrieve ( +WidgetName )
```

xmClipboardStartRetrieve/1 initiates the copying of data from the clipboard. While it is doing so the clipboard will be locked, until xmClipboardEndRetrieve/1 is called.

This predicate will not normally be used, since xmClipboardRetrieve/4 always retrieves all the data elements as a single unit, thus making the lock unnecessary.

WidgetName is any widget of the application, but should be the same one as specified in the associated call of xmClipboardStartCopy/3.

Arguments

WidgetName Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

```
environment_error(clipboard_locked)
```

The clipboard has been locked by another application.

Hints

This predicate is not normally required in IF/Prolog, because, unlike in C, the complete data is supplied by xmClipboardRetrieve/4.

See also

IF/Prolog xmClipboardEndRetrieve/1, xmClipboardRetrieve/4
OSF/Motif XmClipboardStartRetrieve()
Delete last data element from the clipboard store

xmClipboardUndoCopy/1 cancels the last data copying action into the clipboard, provided that this was for the same widget, WidgetName. The situation is then restored as it was before this data copy was carried out.

Arguments

WidgetName Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

environment_error(clipboard_locked)
The clipboard has been locked by another application.

See also

OSF/Motif XmClipboardUndoCopy()
Remove lock from clipboard

\[ \text{xmClipboardUnlock} \left( +\text{WidgetName}, +\text{RemoveAllLocks} \right) \]

xmClipboardUnlock/2 removes the lock imposed on the clipboard by the widget called \textit{WidgetName}. Other applications can then access it, provided that there are no other locks in effect. \textit{RemoveAllLocks} specifies whether all locks are to be removed (true) or only one (false).

All the locks can be removed at once:

\texttt{xmClipboardUnlock(WidgetName, true)}

or one by one:

\texttt{xmClipboardUnlock(WidgetName, false)}

### Arguments

- \texttt{WidgetName} Atom, the name of a widget
- \texttt{RemoveAllLocks} Atom: true | false | yes | no | on | off

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\texttt{environment\_error(clipboard\_fail)}

It was not possible to successfully execute the predicate.

### Hints

This predicate is not normally required in IF/Prolog, because, unlike in C, the complete data is supplied by \texttt{xmClipboardRetrieve/4}.

### See also

- IF/Prolog \texttt{xmClipboardLock/1}
- OSF/Motif \texttt{XmClipboardUnlock()}
xmClipbordWithdrawFormat/2

**Cancel outstanding data copying operations**

| xmClipbordWithdrawFormat ( +WidgetName, +DataId ) |

xmClipbordWithdrawFormat/2 withdraws the notification of a delayed data copying operation. This should be done, at the latest, before the application is terminated. xmClipbordInquirePendingItems/3 returns all notifications of delayed data copying operations.

*WidgetName* is any widget of the application, but should be the same one as specified in the associated call of xmClipbordStartCopy/3.

*DataId* is the identifier used to notify a delayed data copying action. This is returned by xmClipbordCopyName/7). This identifier becomes invalid after *ClipboardPredicate* has been called, or after the call of xmClipbordWithdrawFormat/2, and may then no longer be used.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>DataId</td>
<td>Opaque</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

```environment_error(clipboard_locked)`
The clipboard has been locked by another application.```

**Hints**

Following the call of xmClipbordWithdrawFormat/2, DataId becomes invalid and may no longer be used.

**See also**

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xmClipbordCopyByName/5, xmClipbordCopyName/7</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF/Motif</td>
<td>XmClipbordWithdrawFormat()</td>
</tr>
</tbody>
</table>
Append text to the Command widget command

xmCommandAppendValue(+WidgetName, +Command)

xmCommandAppendValue/2 appends the text passed with Command to the contents of the command area of the Command widget called WidgetName.

Arguments

WidgetName: Atom, the name of a Command widget
Command: Atom or list, a valid XmString

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog: xmCommandSetValue/2
OSF/Motif: XmCommandAppendValue()
xmCommandError/2

Output error message to the Command widget history area

```
xmCommandError ( +WidgetName, +ErrorMessage )
```

xmCommandError/2 outputs an error message to the history area of the Command widget called \textit{WidgetName}. The string \textit{ErrorMessage} is displayed until the next command entry is output.

**Arguments**

- \textit{WidgetName} Atom, the name of a Command widget
- \textit{ErrorMessage} Atom or list, a valid \texttt{XmString}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif \texttt{XmCommandError()}

---

IF/Prolog V5.2

Motif Manual
Query a component of the Command widget

xmCommandGetChild/3 returns the widget name \textit{Child} of a child of the Command widget called \textit{WidgetName}, using the criteria specified by \textit{ChildSpec}. This is used to enable one of the three components of the Command widget, namely command entry, prompt or history list, to be accessed.

Arguments

\begin{itemize}
  \item \textbf{WidgetName}: Atom, the name of a Command widget
  \item \textbf{ChildSpec}: Atom, valid values:
    \begin{itemize}
      \item \texttt{command_text}: command entries
      \item \texttt{prompt_label}: input requests
      \item \texttt{history_list}: history list
      \item \texttt{work_area}: work area
    \end{itemize}
  \item \textbf{Child}: Atom, the name of a widget
\end{itemize}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page \pageref{exceptions}).

Compatibility

\texttt{childSpec} argument value \texttt{work_area} is not available in the interface to OSF/Motif 1.1.

See also

OSF/Motif \quad XmCommandGetChild()
Overwrite a command in the Command widget command list

xmCommandSetValue/2 overwrites the string which is stored in the command area of the Command widget called WidgetName with the string passed with Command.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a Command widget</td>
</tr>
<tr>
<td>Command</td>
<td>Atom or list, a valid XmString</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog: xmCommandAppendValue/2
- OSF/Motif: XmCommandSetValue()
xmConvertUnits/6 converts a value of one unit into the corresponding value of another unit. Dimensions and spacings for a widget can be specified in various units. The units may be: pixels, 100ths of a millimeter, 1000ths of an inch, 100ths of a point, 100ths of a font unit. For example, xmConvertUnits/6 can convert the value 10 measured in pixels to the corresponding value expressed as 100ths of a millimeter.

FromValue is the starting value, expressed in FromUnit units, and this is converted to the value ToValue in ToUnit units.

The dimension of the screen depends on the resolution of the display. A pixel may have a different size in the horizontal and vertical directions. Orientation specifies which direction is to be used when the units are converted.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Orientation</td>
<td>Atom: horizontal</td>
</tr>
<tr>
<td>FromUnit</td>
<td>Atom: pixels</td>
</tr>
<tr>
<td>FromValue</td>
<td>Integer</td>
</tr>
<tr>
<td>ToUnit</td>
<td>Atom: pixels</td>
</tr>
<tr>
<td>ToValue</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

OSF/Motif       XmConvertUnits()
Create widgets

\[
\text{xmCreate} \ ( \ ?\text{WidgetName}, +\text{ParentName}, +\text{ToolkitName}, +\text{ArgList} )
\]

xmCreate.../4 creates a widget of the specified type.

\text{WidgetName} defines the name under which the widget is known in IF/Prolog. If \text{WidgetName} is a variable, then IF/Prolog assigns a unique name and unifies \text{WidgetName} with this name.

\text{ParentName} specifies the widget which is the parent of the widget \text{WidgetName}.

\text{ToolkitName} defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as \text{WidgetName}.

\text{ArgList} contains the resource definitions which are to apply for the \text{WidgetName} widget.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{WidgetName}</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>\text{ParentName}</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>\text{ToolkitName}</td>
<td>Atom</td>
</tr>
<tr>
<td>\text{ArgList}</td>
<td>A list of value assignments for resources.</td>
</tr>
<tr>
<td></td>
<td>These value assignments have the form: {\text{ResourceName} = \text{ResourceValue}, \ldots} or {\text{ResourceName}(\text{ResourceValue}), \ldots}</td>
</tr>
</tbody>
</table>

The following predicates create widgets or gadgets of particular types:

- \text{xmCreateArrowButton/4}
- \text{xmCreateArrowButtonGadget/4}
- \text{xmCreateBulletinBoard/4}
- \text{xmCreateBulletinBoardDialog/4}
- \text{xmCreateCascadeButton/4}
- \text{xmCreateCascadeButtonGadget/4}
- \text{xmCreateCommand/4}
- \text{xmCreateDialogShell/4}
- \text{xmCreateDrawingArea/4}
- \text{xmCreateDrawnButton/4}
- \text{xmCreateErrorDialog/4}
- \text{xmCreateFileChooser/4}
- \text{xmCreateFileChooserDialog/4}
- \text{xmCreateForm/4}
- \text{xmCreateFormDialog/4}
- \text{xmCreateFrame/4}
- \text{xmCreateInformationDialog/4}
- \text{xmCreatePushButton/4}
- \text{xmCreatePushButtonGadget/4}
- \text{xmCreateQuestionDialog/4}
- \text{xmCreateRadioBox/4}
- \text{xmCreateRowColumn/4}
- \text{xmCreateScale/4}
- \text{xmCreateScrollBar/4}
- \text{xmCreateScrollList/4}
- \text{xmCreateScrolledList/4}
- \text{xmCreateScrolledText/4}
- \text{xmCreateScrolledWindow/4}
- \text{xmCreateSelectionBox/4}
- \text{xmCreateSelectionDialog/4}
- \text{xmCreateSeparator/4}
- \text{xmCreateSeparatorGadget/4}
- \text{xmCreateSiHelp/4}
- \text{xmCreateSiHelpDialog/4}
- \text{xmCreateSimpleCheckBox/4}
### Predicates

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmCreateLabel/4</code></td>
<td><code>xmCreateSimpleMenuBar/4</code></td>
</tr>
<tr>
<td><code>xmCreateLabelGadget/4</code></td>
<td><code>xmCreateSimpleOptionMenu/4</code></td>
</tr>
<tr>
<td><code>xmCreateList/4</code></td>
<td><code>xmCreateSimplePopupMenu/4</code></td>
</tr>
<tr>
<td><code>xmCreateMainWindow/4</code></td>
<td><code>xmCreateSimplePulldownMenu/4</code></td>
</tr>
<tr>
<td><code>xmCreateMenuBar/4</code></td>
<td><code>xmCreateSimpleRadioBox/4</code></td>
</tr>
<tr>
<td><code>xmCreateMenuShell/4</code></td>
<td><code>xmCreateText/4</code></td>
</tr>
<tr>
<td><code>xmCreateMessageBox/4</code></td>
<td><code>xmCreateTextField/4</code></td>
</tr>
<tr>
<td><code>xmCreateMessageDialog/4</code></td>
<td><code>xmCreateToggleButton/4</code></td>
</tr>
<tr>
<td><code>xmCreateOptionMenu/4</code></td>
<td><code>xmCreateToggleButtonGadget/4</code></td>
</tr>
<tr>
<td><code>xmCreatePanedWindow/4</code></td>
<td><code>xmCreateWarningDialog/4</code></td>
</tr>
<tr>
<td><code>xmCreatePopupMenu/4</code></td>
<td><code>xmCreateWorkArea/4</code></td>
</tr>
<tr>
<td><code>xmCreatePromptDialog/4</code></td>
<td><code>xmCreateWorkingDialog/4</code></td>
</tr>
</tbody>
</table>

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

### See also

- OSF/Motif
- `XmCreateArrowButton()`, `XmCreate...()`
xmCreateDragIcon/4 creates a DragIcon widget with the name ToolkitName. WidgetName defines the name under which the widget is known in IF/Prolog. If WidgetName is a variable, then IF/Prolog assigns a unique name and unifies WidgetName with this name. ToolkitName defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as WidgetName. ArgList contains the resource definitions which are to apply for the WidgetName widget. AttributeWidget is the name of any widget of the application. It will be used for default visual attributes of the DragIcon widget WidgetName.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>AttributeWidget</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ToolkitName</td>
<td>Atom</td>
</tr>
<tr>
<td>ArgList</td>
<td>A list of value assignments for resources. These value assignments have the form: [ResourceName = ResourceValue, ...] or [ResourceName(ResourceValue), ...]</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Example

... xtGetValues(toplevel, [foreground(FG), background(BG)]), xmGetPixmapByDepth('/.../pixmaps/file.xbm', FG, BG, 1, Icon), xmCreateDragIcon(Drag_icon, Widget, Drag_icon, [ pixmap(Icon) ]), ...

Compatibility

xmCreateDragIcon/4 is not available in the interface to OSF/Motif 1.1.
See also

OSF/Motif      XmCreateDragIcon()
Convert character string

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmCvtCTToXmString</code></td>
<td>(+CompoundText, ?CompoundString)</td>
</tr>
<tr>
<td><code>xmCvtXmStringToCT</code></td>
<td>(+CompoundString, ?CompoundText)</td>
</tr>
</tbody>
</table>

These predicates convert compound strings to a compound text format and vice versa. Compound strings are arbitrary combinations of character strings, font names, line separators and write directions. This enables multi-line character strings to be output using different fonts and written in different directions.

`xmCvtCTToXmString/2` converts a compound text to a compound string.

`xmCvtXmStringToCT/2` converts a compound string to a compound text.

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Arguments

- **CompoundText**: Atom, conforming to the X convention
- **CompoundString**: Atom or list, a valid XmString

See also

- OSF/Motif: XmCvtXmStringToCT()
Deactivate protocols

```
xmDeactivateProtocol ( +ShellName, +Property, +Protocol )
xmDeactivateWMProtocol ( +ShellName, +Protocol )
```

These predicates deactivate protocols for inter-client communication.

xmDeactivateProtocol/3 deactivates the Protocol Manager protocol specified by `Protocol` and removes it from the protocol list which is linked under the name `Property` with the Shell widget `ShellName`.

xmDeactivateWMProtocol/2 deactivates the Protocol Manager protocol specified by `Protocol` and removes it from the predefined protocol list, WM_PROTOCOLS, of the Shell widget `ShellName`.

As long as a protocol is deactivated, no messages can be received for the protocol.

**Arguments**

- **ShellName**: Atom, the name of a widget
- **Property**: Atom
- **Protocol**: Atom

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmActivateProtocol/3`, `xmActivateWMProtocol/2`, `xmSendClientMessage/5`
- OSF/Motif: `XmDeactivateProtocol()`, ...WMProtocol()
Abort drag operation

\[
\text{xmDragCancel} \ ( +\text{DragContextWidget} )
\]

\(\text{xmDragCancel/1}\) terminates the drag operation and cancels any pending actions of the Drag Context identified by \textit{DragContextWidget}.

**Arguments**

\begin{itemize}
  \item \text{DragContextWidget} \quad \text{Atom, the name of a widget}
\end{itemize}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

\(\text{xmDragCancel/1}\) is not available in the interface to OSF/Motif 1.1.

**See also**

\begin{itemize}
  \item IF/Prolog \quad \text{xmDragStart/3}
  \item OSF/Motif \quad \text{XmDragCancel()}
\end{itemize}
**Initiate a drag operation**

```
xmDragStart( ?DragContextWidget, +EnclosingWidget, +ArgList )
```

`xmDragStart/3` initiates a drag operation and returns the DragContext widget.

*DragContextWidget* defines the name under which the widget is known in IF/Prolog. If *DragContextWidget* is a variable, then IF/Prolog assigns a unique name and unifies *DragContextWidget* with this name.

*EnclosingWidget* must be the smallest widget that encloses all source elements for the drag operation.

*ArgList* contains the resource definitions which are to apply for the *WidgetName* widget.

**Arguments**

- **DragContextWidget**: Atom, the name of a widget
- **EnclosingWidget**: Atom, the name of an enclosing widget
- **ArgList**: A list of value assignments for resources.
  These value assignments have the form: `[ResourceName = ResourceValue, ...]` or `[ResourceName(ResourceValue), ...]

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **system_error(no_event_to_analyze)**
  The `xtGetEventComponent(s)/1` predicate may only be called from within a callback.

- **system_error(button_press_event_expected)**
  The event that triggered the drag operation was not a Button-Press event as it should be.

**Example**

```
...,
  xmDragStart(Drag, Widget, [blendModel(blend_just_source),
   cursorBackground(BG),
   sourceCursorIcon(Drag_icon),
   exportTargets(['FILE_CONTENTS', 'FILE_NAME', 'STRING']),
   dragOperations([drop_copy])],
...)
```
xmDragStart/3

\begin{verbatim}
  clientData(File),
  convertProc(convert_proc)
\end{verbatim}

...

**Compatibility**

xmDragStart/3 is not available in the interface to OSF/Motif 1.1.

**See also**

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xmDragCancel/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF/Motif</td>
<td>XmDragStart()</td>
</tr>
</tbody>
</table>
Reorder stack of widgets registered as drop sites

xmDropSiteConfigureStackingOrder/3 changes the stacking order of the drop site specified by DropSiteWidget. This determines the way drag-under effects are clipped by overlapping siblings.

DropSiteWidget is a widget that has been either registered as a drop site using xmDropSiteRegister/2 or has built in drop site functionality, such as a text widget.

SiblingDropSite specifies a sibling widget relative to which the drop sites are restacked.

StackMode specifies the new stack position for DropSiteWidget relative to the SiblingDropSite widget.

Arguments

DropSiteWidget Atom, the name a widget registered as a drop site
SiblingDropSite Atom, the name a sibling widget registered as drop site
StackMode Atom: above | below

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error(stack_mode)

The stack mode must be specified as one of the atoms: above or below.

Compatibility

xmDropSiteConfigureStackingOrder/3 is not available in the interface to OSF/Motif 1.1.

See also

IF/Prolog xmDropSiteEndUpdate/1, xmDropSiteQueryStackingOrder/3, xmDropSiteRegister/3, xmDropSiteRetrieve/2, xmDropSiteStartUpdate/1, xmDropSiteUpdate/2, xmDropTransferStart/3
OSF/Motif XmDropSiteConfigureStackingOrder()
Economize multiple calls to `xmDropSiteUpdate`

```
xmDropSiteEndUpdate ( +DropSiteWidget )
```

`xmDropSiteEndUpdate/1` terminates a sequence of multiple calls to `xmDropSiteUpdate/2` that must be started with a call to `xmDropSiteStartUpdate/1`.

**Arguments**

- **DropSiteWidget** 
  Atom, the name a widget registered as a drop site

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

`xmDropSiteEndUpdate/1` is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog
  - `xmDropSiteConfigureStackingOrder/3`, `xmDropSiteQueryStackingOrder/3`, `xmDropSiteRegister/3`, `xmDropSiteRetrieve/2`, `xmDropSiteStartUpdate/1`, `xmDropSiteUpdate/2`, `xmDropTransferStart/3`
- OSF/Motif
  - `XmDropSiteEndUpdate()`
Retrieve drop site parent and drop site children of widget

\[
\text{xmDropSiteQueryStackingOrder}(\ +\text{DropSiteWidget}, \ ?\text{DropSiteParent}, \ ?\text{DropSiteChildren})
\]

xmDropSiteQueryStackingOrder/3 queries the stacking order of the \text{DropSiteWidget}'s children \text{DropSiteChildren} registered as drop sites and the drop site parent \text{DropSiteParent}.

\text{DropSiteWidget} is a widget that has been either registered as a drop site using xmDropSiteRegister/2 or has built in drop site functionality, such as a text widget.

**Arguments**

- DropSiteWidget Atom, the name a widget registered as a drop site
- DropSiteParent Atom, the name a parent widget registered as a drop site
- DropSiteChildren List of atoms, child widgets registered as drop sites

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmDropSiteQueryStackingOrder/3 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog xmDropSiteConfigureStackingOrder/3, xmDropSiteEndUpdate/1, xmDropSiteRegister/3, xmDropSiteRetrieve/2, xmDropSiteStartUpdate/1, xmDropSiteUpdate/2, xmDropTransferStart/3
- OSF/Motif XmDropSiteQueryStackingOrder()
xmDropSiteRegister/2

Register widget as a drop site

\[
\text{xmDropSiteRegister} \left( +\text{WidgetName}, +\text{ArgList} \right)
\]

xmDropSiteRegister/2 registers \textit{Widget} as a drop site. The resources in \textit{ArgList} define the behaviour of the drop site.

\textit{ArgList} contains the resource definitions which are to apply for the \textit{DropSiteWidget} widget.

\textbf{Arguments}

- \textbf{WidgetName} \hspace{1cm} Atom, the name of a widget
- \textbf{ArgList} \hspace{1cm} A list of value assignments for resources.

These value assignments have the form: \text{[ResourceName = ResourceValue, ...]} or \text{[ResourceName(ResourceValue), ...]}

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{Example}

\[
\ldots
\text{xmDropSiteRegister(filename, [}
\text{importTargets([‘FILE_CONTENTS’, ‘FILE_NAME’])},
\text{dropSiteOperations([drop_copy]),}
\text{dropProc(handle_drop_file_label) ])},
\ldots
\]

\textbf{Compatibility}

xmDropSiteRegister/2 is not available in the interface to OSF/Motif 1.1.

\textbf{See also}

- IF/Prolog \text{xmDropSiteConfigureStackingOrder/3, xmDropSiteEndUpdate/1, xmDropSiteQueryStackingOrder/3, xmDropSiteRetrieve/2, xmDropSiteStartUpdate/1, xmDropSiteUpdate/2, xmDropTransferStart/3}
- OSF/Motif \text{XmDropSiteRegister()}
Retrieves resource values of a drop site

**xmDropSiteRetrieve/2** retrieves resource values of a drop site widget *DropSiteWidget* in a list *ArgList*.

*DropSiteWidget* is a widget that has been either registered as a drop site using *xmDropSiteRegister/2* or has built in drop site functionality, such as a text widget.

*ArgList* contains the resource definitions which are to apply for the *WidgetName* widget.

### Arguments

- **DropSiteWidget**: Atom, the name a widget registered as a drop site
- **ArgList**: A list of value assignments for resources.

These value assignments have the form: `[ResourceName = ResourceValue, ...]` or `[ResourceName(ResourceValue), ...]`

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

### Example

```
...  
xmDropSiteRetrieve(text, [importTargets(ImportTargets),
                        dropProc(DropProc)],
...)
```

### Compatibility

*xmDropSiteRetrieve/2* is not available in the interface to OSF/Motif 1.1.

### See also

- IF/Prolog: *xmDropSiteConfigureStackingOrder/3*, *xmDropSiteEndUpdate/1*, *xmDropSiteQueryStackingOrder/3*, *xmDropSiteRegister/3*, *xmDropSiteStartUpdate/1*, *xmDropSiteUpdate/2*, *xmDropTransferStart/3*
- OSF/Motif: XmDropSiteRetrieve()
Economize multiple calls to xmDropSiteUpdate

**xmDropSiteStartUpdate** ( +WidgetName )

xmDropSiteStartUpdate/1 starts a sequence of multiple calls to xmDropSiteUpdate/2 that must be terminated with a call to xmDropSiteEndUpdate/1.

**Arguments**

- **WidgetName**  Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmDropSiteStartUpdate/1 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog  xmDropSiteConfigureStackingOrder/3, xmDropSiteEndUpdate/1, xmDropSiteQueryStackingOrder/3, xmDropSiteRegister/3, xmDropSiteRetrieve/2, xmDropSiteUpdate/2, xmDropTransferStart/3
- OSF/Motif  XmDropSiteStartUpdate()
Predicates

xmDropSiteUnregister/1

Free drop site information for widget

| xmDropSiteUnregister ( +DropSiteWidget ) |

xmDropSiteUnregister/1 is called to inform the toolkit that \textit{DropSiteWidget} is no longer needed as a drop site.

\textit{DropSiteWidget} is a widget that has been either registered as a drop site using xmDropSiteRegister/2 or has built in drop site functionality, such as a text widget.

Arguments

- \textit{DropSiteWidget} Atom, the name a widget registered as a drop site

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

xmDropSiteUnregister/1 is not available in the interface to OSF/Motif 1.1.

See also

- IF/Prolog \quad \textit{xmDropSiteConfigureStackingOrder/3, xmDropSiteEndUpdate/1, xmDropSiteQueryStackingOrder/3, xmDropSiteRegister/3, xmDropSiteRetrieve/2, xmDropSiteStartUpdate/1, xmDropSiteUpdate/2, xmDropTransferStart/3}
- OSF/Motif \quad \textbf{XmDropSiteUnregister}()
xmDropSiteUpdate/2

Update resources for registered DropSite

```
xmDropSiteUpdate ( +DropSiteWidget, +ArgList )
```

xmDropSiteUpdate/2 updates resources in \textit{ArgList} for a registered drop site widget \textit{DropSiteWidget}.

\textit{ArgList} contains the resource definitions which are to apply for the \textit{DropSiteWidget} widget.

Multiple calls to \textit{xmDropSiteUpdate/2} can be bracketed with \textit{xmDropSiteStartUpdate/1} and \textit{xmDropSiteEndUpdate/1} which should make updating faster.

**Arguments**

\begin{itemize}
  \item \textbf{DropSiteWidget} \hspace{1cm} Atom, the name a widget registered as a drop site
  \item \textbf{ArgList} \hspace{1cm} A list of value assignments for resources.
      These value assignments have the form: [\textit{ResourceName} = \textit{ResourceValue}, ...] or [\textit{ResourceName} (\textit{ResourceValue}), ...]
\end{itemize}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Example**

```
...  
  xmDropSiteUpdate(text, [  
      importTargets(NewImportTargets),  
      dropProc(handle_drop_text)],  
    ...  
```

**Compatibility**

\textit{xmDropSiteUpdate/2} is not available in the interface to OSF/Motif 1.1.

**See also**

\begin{itemize}
  \item IF/Prolog \hspace{1cm} \textit{xmDropSiteConfigureStackingOrder/3}, \textit{xmDropSiteEndUpdate/1}, \textit{xmDropSiteQueryStackingOrder/3}, \textit{xmDropSiteRegister/3}, \textit{xmDropSiteRetrieve/2}, \textit{xmDropSiteStartUpdate/1}, \textit{xmDropTransferStart/3}
  \item OSF/Motif \hspace{1cm} \textit{XmDropSiteUpdate()}
\end{itemize}
Enable processing of additional entries after initiating a drop transfer

```
xmDropTransferAdd ( +DropSiteWidget, +DropTransferEntries )
```

xmDropTransferAdd/2 can be called to register additional drop transfer entries after initiating a drop transfer.

*DropSiteWidget* is a widget that has been either registered as a drop site using `xmDropSiteRegister/2` or has built in drop site functionality, such as a text widget. *Targets* are implicitly converted to OSF/Motif atoms in IF/Prolog.

**Arguments**

- **DropSiteWidget**  
  Atom, the name a widget registered as a drop site
- **DropTransferEntries**  
  Structure or list of structures of the form `entry(Target, ClientData)`
- **Target**  
  Atom, the name of a target atom
- **ClientData**  
  Atom, client data

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmDropTransferAdd/2 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog  
  `xmDropSiteConfigureStackingOrder/3`,  
  `xmDropSiteEndUpdate/1`,  
  `xmDropSiteQueryStackingOrder/3`,  
  `xmDropSiteRegister/3`,  
  `xmDropSiteRetrieve/2`,  
  `xmDropSiteStartUpdate/1`,  
  `xmDropSiteUpdate/2`,  
  `xmDropTransferStart/3`
- OSF/Motif  
  `XmDropTransferAdd()`
**Initiate a drop transfer**

\[
\text{xmDropTransferStart} \left( \text{?WidgetName}, \ +\text{DragContextWidget}, \ +\text{ArgList} \right)
\]

xmDropTransferStart/3 initiates a drop transfer using the specified argument list \textit{ArgList} and the name \textit{WidgetName} as the name of the DropTransfer widget.

\textit{DragContextWidget} is the name of a widget used in a call to \textit{xmDragStart}/3.

\textit{ArgList} contains the resource definitions which are to apply for the \textit{WidgetName} widget.

**Arguments**

- \textit{WidgetName} \hspace{1cm} Atom, the name of a widget
- \textit{DragContextWidget} \hspace{1cm} Atom, the name of a widget
- \textit{ArgList} \hspace{1cm} A list of value assignments for resources. These value assignments have the form: \([\text{ResourceName} = \text{ResourceValue}, \ldots]\) or \([\text{ResourceName}(\text{ResourceValue}), \ldots]\)

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Example**

```prolog
.../*
** Register the filename widget as a drop site.
*/
xmDropSiteRegister(filename, [
    importTargets(["FILE_CONTENTS", "FILE_NAME"]),
    dropSiteOperations([drop_copy]),
    dropProc(handle_drop_file_label)
]),
...

/*
** handle_drop_file_label gets called when something is dropped on
** the filename widget.
*/
handle_drop_file_label(_Widget, _Ca, _Cl) :-
xmGetCallbackComponents([
    dragContext(DragContext),
...
```
Predicates

\[
dropAction(DropAction),
operation(Operation),
dropSiteStatus(_Status)\]
\[
),
xtGetValue(DragContext, exportTargets(ExportTargets)),
handle_drop_file_label(DragContext, DropAction, Operation,
  ExportTargets).
\]

... handle_drop_file_label(DragContext, drop, drop_copy, Export Targets) :-
member('FILE_NAME', ExportTargets),
member('FILE CONTENTS', ExportTargets),
!,
xmDropTransferStart(DropTransfer, DragContext, [
  transferProc(transfer_filename),
  dropTransfers([entry('FILE CONTENTS', gabba),
                 entry('FILE NAME', hey)])
]),
xtAddCallback(DropTransfer, destroyCallback,
  dropTransferDestroyCB, gabba).
...

dropTransferDestroyCB(Widget,_,_) :-
    /*
    ** The widget is automatically destroyed. We have to remove
    ** the widget from the Prolog widget tree explicitly.
    */
    xtUnregisterWidget(Widget).
...

Compatibility

xmDropTransferStart/3 is not available in the interface to OSF/Motif 1.1.

See also

IF/Prolog

xmDropSiteConfigureStackingOrder/3,
xmDropSiteEndUpdate/1, xmDropSiteQueryStackingOrder/3,
xmDropSiteRegister/3, xmDropSiteRetrieve/2,
xmDropSiteStartUpdate/1, xmDropSiteUpdate/2

OSF/Motif

XmDropTransferStart()
Retrieve components of the FileSelectionBox widget

```
xmFileSelectionBoxGetChild( +WidgetName, +ChildSpec, ?Child )
```

`xmFileSelectionBoxGetChild/3` returns the widget name `Child` for a child of the FileSelectionBox widget `WidgetName`. A FileSelectionBox is a widget which consists of a number of components. This predicate is used to access the `ChildSpec` component of the widget.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a FileSelectionBox widget</td>
</tr>
<tr>
<td>ChildSpec</td>
<td>Atom, valid values:</td>
</tr>
<tr>
<td></td>
<td><code>apply_button</code></td>
</tr>
<tr>
<td></td>
<td><code>cancel_button</code></td>
</tr>
<tr>
<td></td>
<td><code>default_button</code></td>
</tr>
<tr>
<td></td>
<td><code>dir_list, dir_list_label</code></td>
</tr>
<tr>
<td></td>
<td><code>help_button</code></td>
</tr>
<tr>
<td></td>
<td><code>filter_label</code></td>
</tr>
<tr>
<td></td>
<td><code>filter_text</code></td>
</tr>
<tr>
<td></td>
<td><code>list</code></td>
</tr>
<tr>
<td></td>
<td><code>list_label</code></td>
</tr>
<tr>
<td></td>
<td><code>ok_button</code></td>
</tr>
<tr>
<td></td>
<td><code>selection_label</code></td>
</tr>
<tr>
<td></td>
<td><code>separator</code></td>
</tr>
<tr>
<td></td>
<td><code>text</code></td>
</tr>
<tr>
<td></td>
<td><code>work_area</code></td>
</tr>
<tr>
<td>Child</td>
<td>Atom, the name of a widget</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif: `XmFileSelectionBoxDoSearch()`
Search file system for a directory or file

\[
\text{xmFileSelectionDoSearch} \left( +\text{WidgetName}, +\text{DirMask} \right)
\]

xmFileSelectionDoSearch/2 causes a search to be made for a directory or a file in a file system, using a specified mask \textit{DirMask} in the FileSelectionBox widget named \textit{WidgetName}.

**Arguments**

- **WidgetItem**: Atom, the name of a FileSelectionBox widget
- **DirMask**: Atom or list, a valid XmString

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif
- XmFileSelectionBoxDoSearch()
Append entries to a Font List

xmFontListAppend/3 appends new font list entries from the list Entries and to a font list FontList and returns a new font list NewFontList with the additional entries. FontListEntries is an font list entry or a list of font list entries. Each entry consists of a tag and a font specification. If a font set is specified the font list type must be set to is_font_set.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontList</td>
<td>Opaque, the name of a font list</td>
</tr>
<tr>
<td>FontListEntries</td>
<td>Structure or list of structures of the form entry(FontListName, FontListType, FontListTag)</td>
</tr>
<tr>
<td>FontListName</td>
<td>Atom, the name of a font list or font set</td>
</tr>
<tr>
<td>FontListType</td>
<td>Atom, is_font</td>
</tr>
<tr>
<td>FontListTag</td>
<td>Atom, the name of a font list tag</td>
</tr>
<tr>
<td>NewFontList</td>
<td>Opaque, the name of a font list</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error(XmFontList)
    The specified argument must be an identifier for a font list.
domain_error(font_list_entry)
    The specified argument must be a font list entry structure of the form: entry(name, type, tag).
domain_error(font_list_type)
    The specified argument must be a type identifier for a font list entry: is_font or is_font_set.

Example

...xmFontListAppend(FontList, entry('**-times-*--*-r-**-120-**', is_font, tag4), NewFontList),...

IF/Prolog V5.2 267 Motif Manual
Predicates

xmFontListAppend/3

Compatibility

xmFontListAppend/3 is not available in the interface to OSF/Motif 1.1.

See also

IF/Prolog  xmFontListFree/1, xmFontListCopy/2, xmFontListCreate/2
OSF/Motif  XmFontList, XmFontListAdd(), XmFontListAppendEntry(),
            XmFontListCreate(), XmFontListEntryCreate()
Copy a Font List

\textbf{xmFontListCopy} (~ +FontList, -FontListCopy ~)

\textit{xmFontListCopy/2} copies the font list \textit{FontList} and returns a new font list \textit{FontListCopy}.

\textbf{Arguments}

- \textit{FontList} : Opaque, the name of a font list
- \textit{FontListCopy} : Opaque, the name of a font list

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textit{domain_error(XmFontList)}

The specified argument must be an identifier for a font list.

\textbf{Compatibility}

\textit{xmFontListCopy/2} is not available in the interface to OSF/Motif 1.1.

\textbf{See also}

- IF/Prolog \hspace{1cm} \textit{xmFontListAppend/3, xmFontListFree/1, xmFontListCreate/2}
- OSF/Motif \hspace{1cm} \textit{XmFontListCopy()}
Create a new FontList from entry or list of entries

```
xmFontListCreate ( +FontListEntries, -FontList )
```

xmFontListCreate/2 creates a new font list `FontList` from a list of font list entries `FontListEntries`.

**Arguments**

- **FontListEntries**: Structure or list of structures of the form `entry(FontListName, FontListType, FontListTag)`
- **FontListName**: Atom, the name of a font list or font set
- **FontListType**: Atom, `is_font` | `is_font_set`
- **FontListTag**: Atom, the name of a font list tag
- **FontList**: Opaque, the name of a font list

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(XmFontList)**
  The specified argument must be an identifier for a font list.
- **domain_error(font_list_entry)**
  The specified argument must be a font list entry structure of the form: `entry(name, type, tag)`.
- **domain_error(font_list_type)**
  The specified argument must be a type identifier for a font list entry: `is_font` or `is_font_set`.

**Example**

```
...xmFontListCreate([
  entry('*-courier-*-r-****-120-*', is_font, tag1),
  entry('*-courier-bold-o-***-120-*', is_font, tag2),
  entry('*-courier-medium-r-****-120-*', is_font, tag3)],
  FontList),
...`
```
xmFontListCreate/2

Compatibility

xmFontListCreate/2 is not available in the interface to OSF/Motif 1.1.

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>OSF/Motif</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmFontListAppend/3, xmFontListFree/1, xmFontListCopy/2</td>
<td>XmFontList, XmFontListAdd(), XmFontListAppendEntry(), XmFontListCreate(), XmFontListEntryCreate()</td>
</tr>
</tbody>
</table>

IF/Prolog V5.2 271 Motif Manual
xmFontListFree/1 frees the font list 

\textit{FontList} and recovers all memory.

**Arguments**

\textbf{FontList} \hspace{1cm} \text{Opaque, the name of a font list}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\texttt{domain\_error(XmFontList)}

The specified argument must be an identifier for a font list.

**Compatibility**

\texttt{xmFontListFree/1} is not available in the interface to OSF/Motif 1.1.

**See also**

IF/Prolog \hspace{1cm} \texttt{xmFontListAppend/3, xmFontListCopy/2, xmFontListCreate/2}

OSF/Motif \hspace{1cm} \texttt{XmFontListFree()}

\textbf{Free memory for a Font List}
xmGetAtomName/2 retrieves the name of an atom which is registered with the X server under the identifier \textit{AtomId}. It makes no difference which application registered the atom with the X server.

**Arguments**

\begin{align*}
\text{AtomId} & \quad \text{Integer} \geq 0 \\
\text{AtomName} & \quad \text{Atom}
\end{align*}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

X \quad \text{XGetAtomName()}

Retrieve components of a callback structure

**xmGetCallbackComponent** ( ?Component )

**xmGetCallbackComponents** ( ?CallbackComponentList )

These predicates return the values of callback components.

xmGetCallbackComponent/1 retrieves the component specified by Component from the callback structure for the current callback.

xmGetCallbackComponents/1 retrieves the components from the list named CallbackComponentList.

*CallbackComponentList* is a list of elements in the form:

Component = Value or Component(Value).

Here, Component specifies which component is to be interrogated. Value is unified with the component value. Atoms which correspond to the component names in the C structure can be specified for Component. Where the same component name has multiple uses in Motif, the specifications will no longer match the names. For example, the name "value" is used both for integer and for XmString values. For this reason, a distinction is made in IF/Prolog between the specifications value and valueString.

An exception will result if this predicate is not called from a callback predicate. Exceptions also result if the requested component is not contained in the current callback structure. For this reason, the application must first inspect the reasons for generation of the callback, and then determine what components are valid.

The special component specification ’widget’ queries the callback widget.

If the component is specified as callbackstruct, then Value is not a single variable. It must then be a list like CallbackComponentList.

The event component can also be interrogated. To do so, the application should first check the eventType. The permissible event structure components depend on this. In IF/Prolog, event components are specified as the names of the C structure components prefixed by ‘event’. In this case the start of each new word is indicated by an uppercase letter, not by an underline.

(x -> eventX, in_out_detail -> eventInOutDetail).

The appendix contains a list of all the valid callback and event structure components.

**Arguments**

CallbackComponentList A list of value assignments for components of callback structures

Component Value assignment for components of callback structures
Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**domain_error(component_list)**
The elements specified in the list must be value assignments.

**system_error(no_callback_struct_to_analyze)**
The predicate xmGetCallbackComponent(s)/1 may only be called from within a callback.

**domain_error(class_component)**
The specified component is not allowed for this callback structure, it is not an element of the callback structure of the calling widget.

**domain_error(reason_component)**
The specified component is not allowed for this callback structure.

**domain_error(callback_component)**
The specified callback component is not known to the interface.
Determine destination address for cut and paste functions

\[ \text{xmGetDestination( ?WidgetName )} \]

xmGetDestination/1 returns \textit{WidgetName} for the widget which is currently being used as the output widget on the display. This will generally be the widget on which the last marking, editing or insertion access was made, or a cut and paste access. This address will be used as the destination address for the rapid copy and insert procedure of the cut and paste function.

If xmGetDestination/1 is called before the corresponding access has been executed, \textit{WidgetName} will be unified with \texttt{none}.

**Arguments**

\begin{itemize}
  \item \textit{WidgetName} \quad \text{Atom, the name of a widget}
\end{itemize}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

\begin{itemize}
  \item OSF/Motif \quad \texttt{XmGetDestination()}
\end{itemize}
Return widget name of DragContext active at TimeStamp

```
xmGetDragContext ( +RefWidget, +TimeStamp, ?DragContextWidget )
```

xmGetDragContext/3 returns the name DragContextWidget of the active drag context. RefWidget and TimeStamp serve to uniquely identify the display and the drag context active at time TimeStamp.

TimeStamp is available as a callback component of drop site callback structures using xmGetCallbackComponent(s)/1.

**Arguments**

- RefWidget: Atom, the name of a reference widget
- TimeStamp: Opaque, a time stamp
- DragContextWidget: Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmGetDragContext/3 is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif: XmGetDragContext()
Predicates

xmGetFocusWidget/2

Determine focus widget within hierarchy of widget

```
xmGetFocusWidget ( +RefWidget, ?FocusWidget )
```

xmGetFocusWidget/2 returns the widget FocusWidget that has the keyboard focus within a widget hierarchy.

RefWidget serves to determine the widget hierarchy and can be any widget within the same widget hierarchy.

**Arguments**

- RefWidget Atom, the name of a reference widget
- FocusWidget Atom, the name of a focus widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmGetFocusWidget/2 is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif XmGetFocusWidget()
xmGetMenuCursor/1 returns the id of the pointer $Cursor$ currently in use when a menu is output.

**Arguments**

$Cursor$  
Atom, a valid cursor name

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

IF/Prolog     xmSetMenuCursor/1
OSF/Motif     XmGetMenuCursor()
Generate a pixmap

\texttt{xmGetPixmap( +ImageName, +Foreground, +Background, ?Pixmap )}

\textit{xmGetPixmap/4} uses \textit{ImageName}, \textit{Foreground} and \textit{Background} to look for the pixmap in a pixmap cache. If not found \textit{ImageName} is used as a filename and the pixmap will be created from the information found in this file. The file is looked for in various directories. Please consult the OSF/Motif reference manuals for further details.

**Arguments**

- \textit{ImageName} Atom, the name of a pixmap (file)
- \textit{Foreground} Atom, name of a color
- \textit{Background} Atom, name of a color
- \textit{Pixmap} Opaque, a pixmap

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Example**

```prolog
... xtGetValues(toplevel, [foreground(FG), background(BG)]),
xmGetPixmap('../../pixmaps/file.xbm', FG, BG, Icon),
...```

**Compatibility**

\textit{xmGetPixmap/4} is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog \texttt{xmGetPixmapByDepth/5}
- OSF/Motif \texttt{XmGetPixmap}
xmGetPixmapByDepth/5 uses *ImageName*, *Foreground*, *Background*, and *Depth* to look for the pixmap in a pixmap cache. If not found *ImageName* is used as a filename and the pixmap will be created from the information found in this file. The file is looked for in various directories. Please consult the OSF/Motif reference manuals for further details.

**Arguments**

- **ImageName**: Atom, the name of a pixmap (file)
- **Foreground**: Atom, name of a color
- **Background**: Atom, name of a color
- **Depth**: Integer
- **Pixmap**: Opaque, a pixmap

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

*xmGetPixmapByDepth/5* is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog: xmGetPixmap/4
- OSF/Motif: XmGetPixmapByDepth()
Query menu environment

\textbf{xmGetPostedFromWidget/2}\par

\begin{center}
\textbf{xmGetPostedFromWidget} ( +Menu, ?WidgetName )
\end{center}

\textit{xmGetPostedFromWidget/2} returns the name of the widget \textit{WidgetName}, from within which a callback has been activated to post the menu called \textit{Menu}. An application can use this predicate to determine the environment within which a menu callback is to be interpreted.

\textbf{Arguments}

\begin{itemize}
  \item Menu \hspace{1cm} Atom, the name of a widget of type RowColumn
  \item WidgetName \hspace{1cm} Atom, the name of a widget
\end{itemize}

\textbf{Exceptions} \par

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also} \par

OSF/Motif \hspace{1cm} XmGetPostedFromWidget()
xmGetTabGroup/2

Return tab group widget of widget

\[
\text{xmGetTabGroup} ( \ +\text{WidgetName}, \ ?\text{TabGroupWidget} )
\]

xmGetTabGroup/2 returns the name \textit{TabGroupWidget} of the tab group that contains the widget \textit{WidgetName}.

**Arguments**

- \textbf{WidgetName} Atom, the name of a widget
- \textbf{TabGroupWidget} Atom, the name of a tab group widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page \pageref{exceptions}).

**Compatibility**

\textit{xmGetTabGroup/2} is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif \texttt{XmGetTabGroup()}

Return tear off widget of a menu

\[
\text{xmGetTearOffControl} \ (\ +\text{Menu}, \ ?\text{TearOffWidget} \ )
\]

\text{xmGetTearOffControl/2} can be used to obtain the widget name of the internally created widget that controls the menu \text{Menu} when tearing of a menu.

\text{TearOffWidget} defines the name under which the widget is known in IF/Prolog. If \text{TearOffWidget} is a variable, then IF/Prolog assigns a unique name and unifies \text{TearOffWidget} with this name.

Arguments

- \text{Menu} Atom, the name of a menu widget
- \text{TearOffWidget} Atom, the name of a tear-off widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

\text{xmGetTearOffControl/2} is not available in the interface to OSF/Motif 1.1.

See also

- OSF/Motif  \text{XmGetTearOffControl(\ )}
xmGetVisibility/2

Determine visibility status of a widget with respect to ancestors

\[ \text{xmGetVisibility} \left( +\text{WidgetName}, ?\text{Status} \right) \]

xmGetVisibility/2 determines the visibility status \textit{Status} of the widget \textit{WidgetName}.

**Arguments**

- \text{WidgetName} \hspace{1cm} \text{Atom, the name of a widget}
- \text{Status} \hspace{1cm} \text{Atom: visibility\_unobscured | visibility\_partially\_obscured | visibility\_fully\_obscured}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\text{domain\_error(visibility\_expected)}

The specified argument must be one of the atoms visibility\_unobscured, visibility\_partially\_obscured or visibility\_fully\_obscured.

**Compatibility**

xmGetVisibility/2 is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif \hspace{1cm} XmGetVisibility()
## xmGetXmDisplay/1

Return display widget of display

```
xmGetXmDisplay( ?DisplayWidget )
```

xmGetXmDisplay/1 determines the name `DisplayWidget` of the implicitly created XmDisplay object. `DisplayWidget` serves as a hook for setting display resources.

`DisplayWidget` defines the name under which the widget is known in IF/Prolog. If `DisplayWidget` is a variable, then IF/Prolog assigns a unique name and unifies `DisplayWidget` with this name.

### Arguments

- **DisplayWidget** Atom, the name of the display widget

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

### Compatibility

xmGetXmDisplay/1 is not available in the interface to OSF/Motif 1.1.

### See also

- IF/Prolog  xmGetXmScreen/1
- OSF/Motif  XmGetXmDisplay()
xmGetXmScreen/1

Return screen widget of screen

\[
\text{xmGetXmScreen} ( \ ?\text{ScreenWidget} )
\]

xmGetXmScreen/1 determines the name \textit{ScreenWidget} of the implicitly created XmScreen object. \textit{ScreenWidget} serves as a hook for setting screen resources.

\textit{ScreenWidget} defines the name under which the widget is known in IF/Prolog. If \textit{ScreenWidget} is a variable, then IF/Prolog assigns a unique name and unifies \textit{ScreenWidget} with this name.

Arguments

\text{ScreenWidget} \quad \text{Atom, the name of the screen widget}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

xmGetXmScreen/1 is not available in the interface to OSF/Motif 1.1.

See also

\begin{align*}
\text{IF/Prolog} & \quad \text{xmGetXmDisplay/1} \\
\text{OSF/Motif} & \quad \text{XmGetXmScreen()}
\end{align*}
Register atom with the X server or check its existence

\[
\texttt{xmInternAtom} ( \ +\text{AtomName}, +\text{IfExists}, ?\text{AtomId} )
\]

xmInternAtom/3 registers an atom (character string) with the X server, or checks its existence. Any atom which is registered is available under the identifier \textit{AtomId} to all the applications which are using the X server.

\textit{AtomName} is the character string to be registered or checked.

\textit{IfExists} controls whether the atom is registered or its existence checked. If \textit{IfExists} is \texttt{true}, a check is made on whether \textit{Atom} is registered with the X server. If the atom is registered, the predicate will supply its identifier \textit{AtomId}; otherwise \texttt{xmInternAtom/3} fails to prove its goal. If \textit{IfExists} is \texttt{false}, then the atom will be registered with the X server if it is not already registered.

\textit{AtomId} will be unified with the identifier of the registered atom.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{AtomName}</td>
<td>Atom</td>
</tr>
<tr>
<td>\text{IfExists}</td>
<td>Atom: true</td>
</tr>
<tr>
<td>\text{AtomId}</td>
<td>Integer $\geq 0$</td>
</tr>
</tbody>
</table>
Check status of the Motif Window Manager

```
xmIsMotifWMRunning ( +ShellName )
```

xmIsMotifWMRunning/1 checks whether the Motif Window Manager (mwm) is running for the screen of the shell widget named `ShellName`.

The goal will be successful if the window manager is running, otherwise the predicate will fail.

**Arguments**

- **ShellName**: Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif: XmIsMotifWMRunning()
Identify whether widget can be traversed

\[
\text{xmIsTraversable} \ ( \ +\text{WidgetName} \ )
\]

xmIsTraversable/1 determines whether the widget \textit{WidgetName} is traversable; i.e. can receive focus through keyboard traversal. Please consult the OSF/Motif reference manuals for the conditions of traversability.

**Arguments**

- \textit{WidgetItemName} Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmIsTraversable/1 is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif \texttt{XmIsTraversable()}
### Insert entries in a list

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmListAddItem(+ListWidget, +Item, +Position)</code></td>
<td>Adds the entry <code>Item</code> to the list at the position <code>Position</code>. The predicate checks in addition whether the entry is one of those in the list of marked entries. If it is, the entry will be marked.</td>
</tr>
<tr>
<td><code>xmListAddItems(+ListWidget, +ItemsList, +Position)</code></td>
<td>Adds the entries in the list <code>ItemsList</code> to the list at the position <code>Position</code>. The predicate checks in addition which of the entries is contained in the list of marked entries. Any such entries will be marked.</td>
</tr>
<tr>
<td><code>xmListAddItemUnselected(+ListWidget, +Item, +Position)</code></td>
<td>Adds the entry <code>Item</code> into the list at the position <code>Position</code>. This entry will not be marked, even if <code>Item</code> is contained in the list of marked entries.</td>
</tr>
<tr>
<td><code>xmListAddItemsUnselected(+ListWidget, +ItemsList, +Position)</code></td>
<td>Adds the entries in the list <code>ItemsList</code> to the list at the position <code>Position</code>. The entries will not be marked, even some entry is contained in the list of marked entries.</td>
</tr>
</tbody>
</table>

#### Arguments

- **ListWidget**: Atom, the name of a List widget
- **Item**: Atom or a list, a valid XmString
- **ItemsList**: List of XmStrings
- **Position**: Integer

#### Compatibility

`xmListAddItemsUnselected/3` is not available in the interface to OSF/Motif 1.1.

#### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).
See also

IF/Prolog
xmListDeleteAllItems/1

OSF/Motif
XmListAddItem(), XmListAddItems(),
XmListAddItemUnselected()
Delete entries from a list

xmListDeleteAllItems ( +ListWidget )
xmListDeleteItem ( +ListWidget, +Item )
xmListDeleteItems ( +ListWidget, +ItemsList )
xmListDeleteItemsPos ( +ListWidget, +ItemCount, +Position )
xmListDeletePos ( +ListWidget, +Position )
xmListDeletePositions ( +ListWidget, +PositionList )

These predicates delete entries from the list of $ListWidget$.
xmListDeleteAllItems/1 deletes all the entries from the list.
xmListDeleteItem/2 deletes the entry $Item$ from the list.
xmListDeleteItems/2 deletes the entries in the list $ItemsList$ from the list.
xmListDeleteItemsPos/3 deletes the number of entries given by $ItemCount$ from the list, starting at the position $Position$. The value 1 for $Position$ indicates the first entry in the list.
xmListDeletePos/2 deletes an entry from the list at the position $Position$.
xmListDeletePositions/2 deletes entries from the list at positions in the list $PositionList$.

Arguments

- ListWidget: Atom, the name of a List widget
- Item: Atom or list, a valid XmString
- ItemCount: Integer
- ItemsList: List of valid XmStrings
- Position: Integer
- PositionList: List of integers

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).
Compatibility

xmListDeletePositions/2 is not available in the interface to OSF/Motif 1.1.

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xmListAddItem/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF/Motif</td>
<td>XmListDeleteAllItems(), ...Item(), ...Items(), ...ItemsPos(), ...Pos(), ...Positions()</td>
</tr>
</tbody>
</table>
Cancel markings on entries in a list

- `xmListDeselectAllItems (+ListWidget)`
- `xmListDeselectItem (+ListWidget, +Item)`
- `xmListDeselectPos (+ListWidget, +Position)`

These predicates cancel the markings on entries in the list of `ListWidget` and remove the entries from the list of marked entries.

- `xmListDeselectAllItems/1` cancels the markings on all entries.
- `xmListDeselectItem/2` cancels the marking on the entry `Item`.
- `xmListDeselectPos/3` cancels the marking on the entry `Item` at position `Position`. The value 1 for `Position` indicates the first entry in the list.

**Arguments**

- **ListWidget**: Atom, the name of a List widget
- **Item**: Atom or list, a valid XmString
- **Position**: Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmListSelectItem/3`
- OSF/Motif: XmListDeselectAllItems(), ...Item(), ...Pos()
Query position of a list item

\[
\text{xmListGetKbdItemPos} \ ( \ +\text{ListWidget}, \ ?\text{Position} \ )
\]

xmListGetKbdItemPos/2 queries the position of the list item at the location cursor. The position is unified with \textit{Position}.

**Arguments**

- \textit{ListWidget} \hspace{1cm} \text{Atom, the name of a List widget}
- \textit{Position} \hspace{1cm} \text{Integer}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmListGetKbdItemPos/2 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog \hspace{1cm} \text{xmListSetKbdItemPos/2}
- OSF/Motif \hspace{1cm} \text{XmListGetKbdItemPos()}
Query position of entries in a list

```
xmListGetMatchPos( +ListWidget, +Item, ?PositionList )
```

xmListGetMatchPos/3 returns all positions containing the entry Item in the form of a list named PositionList, or an empty list if Item is not found.

The position of the first entry in the list is 1, the second entry 2, etc.

**Arguments**

- ListWidget: Atom, the name of a List widget
- Item: Atom or list, a valid XmString
- PositionList: List of integers

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xmListGetSelectedPos/2, xmListItemExists/2
- OSF/Motif: XmListGetMatchPos()
Query positions of marked entries in a list

```
xmListGetSelectedPos(+ListWidget, ?PositionList)
```

xmListGetSelectedPos/2 returns all the positions of marked entries in the form of a list PositionList, or an empty list if no marked entries are found.

The position of the first entry in the list is 1, the second entry 2, etc.

**Arguments**

- **ListWidget**: Atom, the name of a List widget
- **PositionList**: List of integers

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmListPosSelected/2, xmListGetMatchPos/3, xmListItemExists/2`
- OSF/Motif: `XmListGetSelectedPos()`
Check existence of an entry in a list

```
 xmListItemExists ( +ListWidget, +Item )
```

xmListItemExists/2 checks whether the entry Item is present in the list of ListWidget. The predicate will fail if Item is not found in the list.

**Arguments**

- ListWidget: Atom, the name of a List widget
- Item: Atom or list, a valid XmString

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xmListGetMatchPos/3, xmListGetSelectedPos/2
- OSF/Motif: XmListItemExists()
Query position of an entry in a list

\[
\text{xmListItemPos ( +ListWidget, +Item, ?Position )}
\]

xmListItemPos/3 returns the Position of the first appearance of the entry Item in the list of ListWidget. The position of the first entry in the list is 1, the second entry 2, etc.

The predicate will fail if the entry Item is not found.

**Arguments**

- ListWidget: Atom, the name of a List widget
- Item: Atom or list, a valid XmString
- Position: Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xmListGetMatchPos/3
- OSF/Motif: XmListItemPos()
Check if list entry is selected

\[
\text{\texttt{xmListPosSelected} ( +ListWidget, +Position )}
\]

\text{xmListPosSelected/2} determines if the list entry at the specified \textit{Position} is selected or not. The predicate succeeds if the list entry is selected; otherwise it fails.

**Arguments**

- ListWidget: Atom, the name of a List widget
- Position: Positive integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

\text{xmListPosSelected/2} is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog: \text{xmListGetSelectedPos/2}, \text{xmListItemPos/3}
- OSF/Motif: XmListPosSelected()
Query bounding box of a list item

\[
\text{xmListPosTo Bounds} \ ( +\text{ListWidget}, +\text{Position}, ?\text{X}, ?\text{Y}, ?\text{Width}, ?\text{Height} )
\]

\text{xmListPosTo Bounds/6} queries the coordinates of an entry within a list and the dimensions of its bounding box. The arguments \( X \) and \( Y \) are unified with the associated \( x \) and \( y \)-coordinates of the upper left corner of the bounding box relative to the upper left corner of the list widget \( \text{ListWidget} \). The arguments \( \text{Width} \) and \( \text{Height} \) are unified with the width and the height of the box.

The predicate fails if the entry at the specified \( \text{Position} \) is not visible.

**Arguments**

- \( \text{ListWidget} \) Atom, the name of a List widget
- \( \text{Position} \) Positive integer
- \( \text{X} \) Integer
- \( \text{Y} \) Integer
- \( \text{Width} \) Integer
- \( \text{Height} \) Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

\text{xmListPosTo Bounds/6} is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog \text{xmListItemPos/3}
- OSF/Motif \text{XmListPosTo Bounds()}

xmListReplace.../3  Predicates

Replace entries in a list

xmListReplaceItems ( +ListWidget, +ListItemsOld, +ListItemsNew )

xmListReplaceItemsPos ( +ListWidget, +ItemsList, +Position )

xmListReplaceItemsPosUnselected ( +ListWidget, +ItemsList, +Position )

xmListReplaceItemsUnselected ( +ListWidget, +ListItemsOld, +ListItemsNew )

xmListReplacePositions ( +ListWidget, +PositionList, +ItemsList )

These predicates replace entries in the ListWidget’s list.

xmListReplaceItems/3 and xmListReplaceItemsUnselected/3 replace all the entries in the list called ListItemsOld by the corresponding entries in the list called ListItemsNew, i.e. the first entry from ListItemsNew overwrites the first entry in the ListItemsOld etc. xmListReplaceItems/3 checks in addition which of the entries is contained in the list of marked entries. Any such entries will be marked.

xmListReplaceItemsPos/3 and xmListReplaceItemsPosUnselected/3 replace the entries in the list by the entries from ItemsList starting from Position. The value 1 represents the first entry in the list. xmListReplaceItemsPos/3 checks in addition which of the entries is contained in the list of marked entries. Any such entries will be marked.

xmListReplacePositions/3 replaces noncontiguous items in a list. The item at each position specified in PositionList is replaced with the corresponding entry in ItemsList.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListWidget</td>
<td>Atom, the name of a List widget</td>
</tr>
<tr>
<td>ItemsList</td>
<td>List of XmStrings</td>
</tr>
<tr>
<td>ListItemsNew</td>
<td>List of XmStrings</td>
</tr>
<tr>
<td>ListItemsOld</td>
<td>List of XmStrings</td>
</tr>
<tr>
<td>Position</td>
<td>Positive integer</td>
</tr>
<tr>
<td>PositionList</td>
<td>List of positive integers</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).
Predicates

**Compatibility**

`xmListReplaceItemsUnselected/3`, `xmListReplaceItemsPosUnselected/3`, `xmListReplacePositions/3` are not available in the interface to OSF/Motif 1.1.

**See also**

OSF/Motif      XmListReplaceItems(), ...Pos()
Mark an entry in a list

xmListSelectItem ( +ListWidget, +Item, +Notify )

xmListSelectPos ( +ListWidget, +Position, +Notify )

These predicates mark an entry in the ListWidget’s list.

xmListSelectItem/3 marks the entry identified by Item in the list.

xmListSelectPos/3 marks the entry at the position given by Position. If the value given for Position is 1, the first entry in the list will be marked, the value 2 marks the second entry, etc. The value 0 marks the last entry in the list.

Notify specifies whether or not a selection callback is created. If Notify is given the value true the application cannot distinguish whether the marking was called by the user or by one of the predicates.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListWidget</td>
<td>Atom, the name of a List widget</td>
</tr>
<tr>
<td>Item</td>
<td>Atom or list, a valid XmString</td>
</tr>
<tr>
<td>Notify</td>
<td>Atom: true</td>
</tr>
<tr>
<td>Position</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

OSF/Motif      XmListSelectItem(), ...Pos()
Set add mode for inserting entries into a list

\[
\text{xmListSetAddMode} \ ( +\text{ListWidget}, +\text{Mode} )
\]

xmListSetAddMode/2 sets the mode for using the keyboard to select entries from the List-Widget’s list. This will apply if the value of the selectionPolicy resource is set to extended_select.

If \text{Mode} has the value \text{true}, then it is possible to mark a number of entries in the list one after another, by a key depression. If \text{Mode} has the value \text{false}, then it is only possible to mark a single entry. If another entry is then marked, the previous marking will be canceled.

**Arguments**

<table>
<thead>
<tr>
<th>ListWidget</th>
<th>Atom, the name of a List widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Atom: true</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

OSF/Motif \hspace{1cm} XmListSetAddMode()
Specify last visible entry in a list

```
xmListSetBottomItem ( +ListWidget, +Item )
```

```
XmListSetBottomPos ( +ListWidget, +Position )
```

These predicates specify which entry in the *ListWidget*’s list is to be the last visible one.

`xmListSetBottomItem/2` makes the entry *Item* the last visible one in the list.

`xmListSetBottomPos/2` makes the entry at the position given by *Position* the last visible one in the list. The value 1 for *Position* means the first entry in the list, the value 2 means the second entry, etc. The value 0 means the last entry in the list.

**Arguments**

- **ListWidget**
  - Atom, the name of a List widget
- **Item**
  - Atom or list, a valid XmString
- **Position**
  - Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog
  - `xmListSetAddMode/2`, `...HorizPos/2`, `...Item/2`, `...Pos/2`
- OSF/Motif
  - `XmListSetBottomItem()`, `...Pos()`
Specify the position of the horizontal scroll bar

```
xmListSetHorizPos( +ListWidget, +Position )
```

xmListSetHorizPos/2 sets the value of the `value` resource for the ScrollBar widget when it is used as a child of the `ListWidget` widget, i.e. the slider for the horizontal scroll bar is set to the position defined by `Position`. In addition, if the `listSizePolicy` resource of the `ListWidget` widget has the value `constant` or `resize_if_possible` and if the horizontal scroll bar is visible, the predicate updates the area of the list which is visible, using the value of `value`.

**Arguments**

- `ListWidget` Atom, the name of a List widget
- `Position` Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif `XmListSetHorizPos()`
Specify the first visible entry in a list

**xmListSetItem** ( +ListWidget, +Item )

**xmListSetPos** ( +ListWidget, +Position )

These predicates specify which entry is to be the first visible entry in the *ListWidget*'s list. 

`xmListSetItem/2` makes the entry identified by *Item* the first visible one for the list.

`xmListSetPos/2` makes the entry at the position specified by *Position* the first visible one for the list. The value 1 for *Position* means the first entry in the list, the value 2 means the second entry, etc. The value 0 means the last entry in the list.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListWidget</td>
<td>Atom, the name of a List widget</td>
</tr>
<tr>
<td>Item</td>
<td>Atom or a list, a valid XmString</td>
</tr>
<tr>
<td>Position</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmListSetBottomItem/2`
- OSF/Motif: `XmListSetItem(), ...Pos()`
Set location cursor

xmListSetKbdItemPos/2 sets the location cursor at the item specified by `Position`.

Arguments

- **ListWidget**: Atom, the name of a List widget
- **Position**: Integer

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

`xmListSetKbdItemPos/2` is not available in the interface to OSF/Motif 1.1.

See also

- IF/Prolog: `xmListGetKbdItemPos/2`
- OSF/Motif: `XmListSetKbdItemPos()`
Update selected list

xmListUpdateSelectedList/1 updates the list of currently selected items of the list widget \textit{ListWidget}.

\textbf{Arguments}

\begin{itemize}
  \item \texttt{ListWidget} \hspace{1cm} Atom, the name of a List widget
\end{itemize}

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{Compatibility}

xmListUpdateSelectedList/1 is not available in the interface to OSF/Motif 1.1.

\textbf{See also}

IF/Prolog \hspace{1cm} \texttt{xmListDeselect.../1/2, xmListSelect.../3}

OSF/Motif \hspace{1cm} \texttt{XmListUpdateSelectedList()}

Query list item position at y-coordinate

\[
\text{xmListYToPos} \ ( +\text{ListWidget}, +Y, \ ?\text{Position} )
\]

\text{xmListYToPos}/3 queries the position of the item at the given y-coordinate \( Y \) within the list. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) indicates that no item exists at the specified y-coordinate.

**Arguments**

- ListWidget: Atom, the name of a List widget
- Y: Integer
- Position: Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

\text{xmListYToPos}/3 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog: \text{xmListPosToBounds}/6
- OSF/Motif: XmListYToPos()
Specify the names of the separator widgets

**xmMainWindowSep1** (*+MainWindowWidget, ?Sep1*)

**xmMainWindowSep2** (*+MainWindowWidget, ?Sep2*)

**xmMainWindowSep3** (*+MainWindowWidget, ?Sep3*)

These predicates supply the widget names of the separators for *MainWindowWidget*.

The first separator is located between the MenuBar and Command widgets, the second between the Command and ScrolledWindow widgets and the third beneath the MessageBox widget.

The separators are only visible if the showSeparator resource for the MainWindow widget is true.

**Arguments**

- **MainWindowWidget**: Atom, the name of a MainWindow widget
- **Sep1**: Atom, the name of a widget
- **Sep2**: Atom, the name of a widget
- **Sep3**: Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

OSF/Motif  
XmMainWindowSep1(), ...Sep2(), ...Sep3()
Specify the children of the MainWindow widget

```
xmMainWindowSetAreas ( +MainWindowWidget, +MenuBar,          
                      +CommandWindow, +HorizontalScrollBar, +VerticalScrollBar, +WorkRegion )
```

xmMainWindowSetAreas/6 specifies which of the possible children of `MainWindowWidget` will actually be managed. The predicate adds the corresponding widgets to the MainWindow widget. Before the child widgets can be assigned to the MainWindow widget, the MainWindow itself must have been created.

All the child widgets are optional. Any argument can be set to the value `none`, with the effect that it will not be supported.

**Arguments**

- **MainWindowWidget** Atom, the name of a MainWindow widget
- **MenuBar** Atom, the name of a widget
- **CommandWindow** Atom, the name of a widget
- **HorizontalScrollBar** Atom, the name of a widget
- **VerticalScrollBar** Atom, the name of a widget
- **WorkRegion** Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmMainWindowSetAreas()
Position pop-up menu

```
xmMenuPosition ( +Menu )
```

xmMenuPosition/1 positions the menu area of a pop-up menu, *Menu*. To carry out this positioning, use is made of the information about the X and Y coordinates, which is passed in the event which initiates the display of the pop-up menu.

**Arguments**

- **Menu**  
  Atom, the name of a widget of type RowColumn

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmMenuPosition()
Query a component of the MessageBox widget

```
xmMessageBoxGetChild ( +WidgetName, +ChildSpec, ?Child )
```

xmMessageBoxGetChild/3 returns the widget name \textit{Child} of a child of the MessageBox widget \textit{WidgetName}, using the criterion specified by \textit{ChildSpec}.

**Arguments**

- \textbf{WidgetName} \hfill Atom, the name of a MessageBox widget
- \textbf{ChildSpec} \hfill Atom, possible values:
  - cancel\_button
  - default\_button
  - help\_button
  - message\_label
  - ok\_button
  - separator
  - symbol\_label
- \textbf{Child} \hfill Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif \hfill XmMessageBoxGetChild()
Get information about a menu item

xmOptionButtonGadget ( +WidgetName, ?Gadget )

xmOptionLabelGadget ( +WidgetName, ?Gadget )

These predicates supply information about the representation of a menu item. The menu item is a widget of type RowColumn. This is made up of various components, e.g. an OptionButton gadget to give a three-dimensional surround (bevel) to the menu item, and a Label gadget to provide the legend.

xmOptionButtonGadget/2 returns the name of the button Gadget to be used for displaying the menu item WidgetName.

xmOptionLabelGadget/2 returns the name of the label Gadget to be used for displaying the menu item WidgetName.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a RowColumn widget</td>
</tr>
<tr>
<td>Gadget</td>
<td>Atom, the name of a gadget</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

OSF/Motif       XmOptionButton(), ...LabelGadget()
Specify the mode for processing keyboard events

\[
\text{xmProcessTraversal} \ ( \ +\text{WidgetName}, +\text{Direction} )
\]

\( \text{xmProcessTraversal/2} \) specifies the component within a widget hierarchy to which keyboard events are to be passed when a widget in the hierarchy has the input focus.

\textit{WidgetName} identifies the widget within whose widget hierarchy the keyboard events are to be passed on. \textit{Direction} specifies how the events are to be passed on.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Direction</td>
<td>Atom: ( \text{traverse_current} \mid \text{traverse_down} \mid \text{traverse_home} \mid \text{traverse_left} \mid \text{traverse_next} \mid \text{traverse_next_tab_group} \mid \text{traverse_prev} \mid \text{traverse_prev_tab_group} \mid \text{traverse_right} \mid \text{traverse_up} )</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmProcessTraversal()
Remove callback predicate from a protocol

```
xmRemoveProtocolCallback ( +ShellName, +Property, +Protocol, +CallbackPredicate, +ClientData )
```

This predicate removes callback predicates from a protocol.

`xmRemoveProtocolCallback/5` removes the callback predicate `CallbackPredicate` from the internal callback list of the protocol identified by `Protocol` and `Property` of the `ShellName` widget.

The callback predicate identified by `CallbackPredicate` is activated when a client message is received for this protocol.

The `ClientData` argument is passed to the callback predicate when the call is made.

**Arguments**

- **ShellName**: Atom, the name of a widget
- **Property**: Atom
- **Protocol**: Atom
- **CallbackPredicate**: Atom, the functor of a callback predicate with an arity of three
- **ClientData**: Term

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmRemoveProtocols/3, xmRemoveWMProtocolCallback/4`
- OSF/Motif: `XmRemoveProtocolCallback()`
Remove protocols

```
xmRemoveProtocols ( +ShellName, +Property, +ProtocolList )
```

This predicate removes inter-client communication protocols.

`xmRemoveProtocols/3` removes the protocols from `ProtocolList`, a list linked to the Shell widget `ShellName` under the name `Property`.

**Arguments**

- `ShellName` Atom, the name of a widget
- `Property` Atom
- `ProtocolList` A list of atoms

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog `xmAddProtocols/3, xmRemoveWMProtocols/2`
- OSF/Motif `XmRemoveProtocols()`
xmRemoveTabGroup/1

Remove a widget from a tab group

xmRemoveTabGroup ( +TabGroupName )

xmRemoveTabGroup/1 removes the widget *TabGroupName* from the list of tab groups. If the keyboard is used to navigate through a hierarchy of widgets, it is possible to put together widgets from the Manager or xmPrimitive classes to form tab groups. Within any tab group, the arrow keys can be used to move the input focus on to the next widget in the group.

**Arguments**

TabGroupName Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

IF/Prolog xmAddTabGroup/1
OSF/Motif XmRemoveTabGroup()
Remove callback predicate from a WM protocol

```
xmRemoveWMProtocolCallback ( +ShellName, +Protocol, +CallbackPredicate, +ClientData )
```

This predicate removes a callback predicate from a protocol.

`xmRemoveWMProtocolCallback/4` removes the callback predicate `CallbackPredicate` from the internal callback list of the protocol `Protocol` which is registered in the protocol list `WM_PROTOCOLS` of the `ShellName` widget.

The callback predicate specified by `CallbackPredicate` is activated when a client message is received for this protocol.

The `ClientData` argument is passed to the callback predicate when the call is made.

**Arguments**

- **ShellName**: Atom, the name of a widget
- **Protocol**: Atom
- **CallbackPredicate**: Atom, the functor of a callback predicate with an arity of three
- **ClientData**: Term

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmAddWMProtocolCallback/5`, `xmRemoveProtocolCallback/5`
- OSF/Motif: `XmRemoveWMProtocolCallback()`
xmRemoveWMProtocols/2

Remove WM protocols

| xmRemoveWMProtocols ( +ShellName, +ProtocolList ) |

This predicate removes inter-client communication protocols.

xmRemoveWMProtocols/2 removes the protocols named in the ProtocolList from the predefined protocol list WM_PROTOCOLS of the Shell widget named ShellName.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShellName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ProtocolList</td>
<td>A list of atoms</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

<table>
<thead>
<tr>
<th>System</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xmAddWMProtocols/2, xmRemoveProtocols/3</td>
</tr>
<tr>
<td>OSF/Motif</td>
<td>XmRemoveWMProtocols()</td>
</tr>
</tbody>
</table>
Install reverse converter

\[ \text{xmRepTypeAddReverse} \ (\ +\text{RepType} \ ) \]

xmRepTypeAddReverse/1 installs the reverse converter for a previously registered representation type converter identified by RepType.

**Arguments**

- **RepType** Atom, the name of a representation type

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmRepTypeAddReverse/1 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog
  - xmRepTypeGetNameList/3, xmRepTypeGetRecord/2, xmRepTypeGetRegistered/1, xmRepTypeRegister/3, xmRepTypeValidValue/3
- OSF/Motif
  - XmRepTypeAddReverse()
Generate list of values for a representation type

\[
\text{xmRepTypeGetNameList}( +\text{RepType}, +\text{UseUppercaseFormat}, ?\text{ValueNameList} )
\]

\text{xmRepTypeGetNameList/3} returns the list of value names for the representation type identified by \text{RepType} in the list \text{ValueNameList}.

**Arguments**

- \text{RepType} \hspace{1cm} \text{Atom, the name of a representation type}
- \text{UseUppercaseFormat} \hspace{1cm} \text{Atom: true | false | yes | no | on | off}
- \text{ValueNameList} \hspace{1cm} \text{List of representation type names}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

\text{xmRepTypeGetNameList/3} is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog \hspace{1cm} \text{xmRepTypeAddReverse/1, xmRepTypeGetRecord/2,}
  \text{xmRepTypeGetRegistered/1, xmRepTypeRegister/3,}
  \text{xmRepTypeValidValue/3}
- OSF/Motif \hspace{1cm} \text{XmRepTypeGetNameList()}

IF/Prolog V5.2 325 Motif Manual
Generate list of data for a representation type

\[
\text{xmRepTypeGetRecord} \left( \text{RepType}, \text{RepTypeInfo} \right)
\]

\text{xmRepTypeGetRecord/2} returns various information about the representation type identified by \textit{RepType} in the parameter \textit{RepTypeInfo}.

**Arguments**

- \textit{RepType} Atom, the name of a representation type
- \textit{RepTypeInfo} List of the form \([\text{RepTypeInstalled}, [\text{RepTyeValues}]]\)

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

\text{xmRepTypeGetRecord/2} is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog \text{xmRepTypeAddReverse/1, xmRepTypeGetNameList/3, xmRepTypeGetRegistered/1, xmRepTypeRegister/3, xmRepTypeValidValue/3}
- OSF/Motif \text{XmRepTypeGetRecord()}

Motif Manual 326 IF/Prolog V5.2
xmRepTypeGetRegistered/1

Generate list of representation type names

```
xmRepTypeGetRegistered ( ?RepTypeList )
```

xmRepTypeGetRegistered/1 returns a list of all registered representation types in \textit{RepTypeList}.

**Arguments**

- \textit{RepTypeList} 
  
  List of representation types

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page \textit{557}).

**Compatibility**

xmRepTypeGetRegistered/1 is not available in the interface to OSF/Motif 1.1.

**See also**

- \textbf{IF/Prolog} 
  
  \texttt{xmRepTypeAddReverse/1}, \texttt{xmRepTypeGetNameList/3}, \texttt{xmRepTypeGetRecord/2}, \texttt{xmRepTypeRegister/3}, \texttt{xmRepTypeValidValue/3}

- \textbf{OSF/Motif} 
  
  \texttt{XmRepTypeGetRegistered()}
Install converter for RowColumn’s TearOffModel Resource

xmRepTypeInstallTearOffModelConverter/0 installs resource converter for the `tearOffModel` resource of RowColumn widgets.

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmRepTypeInstallTearOffModelConverter/0 is not available in the interface to OSF/-Motif 1.1.

**See also**

IF/Prolog

xmRepTypeAddReverse/1, xmRepTypeGetNameList/3,
xmRepTypeGetRecord/2, xmRepTypeGetRegistered/1,
xmRepTypeRegister/3, xmRepTypeValidValue/3

OSF/Motif

XmRepTypeInstallTearOffModelConverter()
**xmRepTypeRegister/3**

Register a representation type resource

```
xmRepTypeRegister( +RepType, +ValueNameList, +ValueList )
```

xmRepTypeRegister/3 registers a new representation type `RepTypeName` with the values `ValueNameList` and the associated numerical values in `ValueList`.

**Arguments**

- **RepType**: Atom, the name of a representation type
- **ValueNameList**: List of representation type names
- **ValueList**: List of representation type values

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(equal_list_lengths)**
  
The specified argument must be a list of the same length as a list in another argument of this predicate.

**Example**

```
...
xmRepTypeRegister('IF/Prolog', [system, constraints, motif, sql], [1, 2, 3, 4]),
...
```

**Compatibility**

xmRepTypeRegister/3 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog: `xmRepTypeAddReverse/1`, `xmRepTypeGetNameList/3`, `xmRepTypeGetRecord/2`, `xmRepTypeGetRegistered/1`, `xmRepTypeValidValue/3`
- OSF/Motif: `XmRepTypeRegister()`
### Test validity of numerical value for representation type resource

| xmRepTypeValidValue ( +RepType, +Value, +WarningWidget ) |

xmRepTypeValidValue/3 determines whether \( Value \) is a valid value for the representation type \( RepType \). The widget \( WarningWidget \) can be used to specify a warning message.

The predicate fails when \( Value \) is not a valid value for the representation type \( RepType \).

Please consult the OSF/Motif reference manuals for on how to use the \( WarningWidget \).

#### Arguments

- **RepType**: Atom, the name of a representation type
- **Value**: Atom, a representation type value
- **WarningWidget**: Atom, the name of a widget or \texttt{none}

#### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

#### Compatibility

xmRepTypeValidValue/3 is not available in the interface to OSF/Motif 1.1.

#### See also

- **IF/Prolog**: `xmRepTypeAddReverse/1`, `xmRepTypeGetNameList/3`, `xmRepTypeGetRecord/2`, `xmRepTypeGetRegistered/1`, `xmRepTypeRegister/3`
- **OSF/Motif**: `XmRepTypeValidValue()`
Query or set position of the slider along a scale

**xmScaleGetValue ( +WidgetName, ?ScrollValue )**

**xmScaleSetValue ( +WidgetName, +ScrollValue )**

These predicates access the value of the slider on the scale in the `WidgetName` widget. The value of this slider must lie within the range from the minimum value (`minimum` resource) to the maximum value (`maximum` resource) for the scale.

`xmScaleGetValue/2` returns `ScrollValue`, the current value of the slider.

`xmScaleSetValue/2` sets `ScrollValue` as the value of the slider.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of the Scale widget</td>
</tr>
<tr>
<td>ScrollValue</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

OSF/Motif       XmScaleGetValue(), ...SetValue()
Query or set the position of a scroll bar slider

**Predicates**

<table>
<thead>
<tr>
<th>xmScrollBarGetValues/5</th>
<th>xmScrollBarSetValues/5</th>
</tr>
</thead>
</table>

These predicates access the following information about a slider on a scroll bar in the widget **WidgetName**.

- **ScrollValue** is the position of the slider, between the minimum value (minimum resource) and the maximum value (maximum resource).
- **SliderSize** is the size of the slider, in pixels.
- **Increment** is the amount by which the slider position will be incremented when it is moved.
- **PageIncrement** is the amount by which the slider will be moved when a page feed is initiated.

**xmScrollBarGetValues/5** returns the current value of the slider.

**xmScrollBarSetValues/5** sets the value of the slider.

**Arguments**

- **WidgetName** Atom, the name of the ScrollBar widget
- **ScrollValue** Integer
- **SliderSize** Integer
- **Increment** Integer
- **PageIncrement** Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmScrollBarGetValues(), ...SetValues()
Make obscured widget or gadget visible

```
xmScrollVisible( +ScrolledWindowWidget, +DescendantWidget, +LeftRightMargin, +TopBottomMargin )
```

xmScrollVisible/4 can be used to make an obscured descendant `DescendantWidget` of a `ScrolledWindow` widget `ScrolledWindowWidget` visible.

The margins `LeftRightMargin` and `TopBottomMargin` specify the margins between the edges of the widget and the viewport.

**Arguments**

- `ScrolledWindowWidget` Atom, the name of a scrolled window widget
- `DescendantWidget` Atom, the name of a widget
- `LeftRightMargin` Integer ≥ 0
- `TopBottomMargin` Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmScrollVisible/4 is not available in the interface to OSF/Motif 1.1.

**See also**

OSF/Motif       XmScrollVisible()
Add or remove components of the ScrolledWindow widget

Predicates

\[ \text{Predicates} \quad \text{xmScrolledWindowSetAreas/4} \]

**Add or remove components of the ScrolledWindow widget**

\[
\text{xmScrolledWindowSetAreas}(\text{+WidgetName, +HorizontalScrollBar, +VerticalScrollBar, +WorkRegion})
\]

\[ \text{xmScrolledWindowSetAreas/4} \text{ adds or removes the following components in a ScrolledWindow widget specified by} \text{ WidgetName:} \]

\[ \text{WorkRegion, VerticalScrollBar, HorizontalScrollBar} \]

Each of these components is optional. If the name **none** is passed for any component, it will not be supported.

**Arguments**

\begin{align*}
\text{WidgetItem} & \quad \text{Atom, the name of a ScrolledWindow widget} \\
\text{HorizontalScrollBar} & \quad \text{Atom, the name of a widget} \\
\text{VerticalScrollBar} & \quad \text{Atom, the name of a widget} \\
\text{WorkRegion} & \quad \text{Atom, the name of a widget}
\end{align*}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

\[ \text{OSF/Motif \quad XmScrolledWindowSetAreas()} \]
xmSelectionBoxGetChild/3

Query a component of the SelectionBox widget

\[
\text{xmSelectionBoxGetChild} ( \ +\text{WidgetName}, +\text{ChildSpec}, ?\text{Child} )
\]

\(\text{xmSelectionBoxGetChild}\)/3 returns \(\text{Child}\), the widget name of a child of the SelectionBox widget named \(\text{WidgetName}\), using the criterion specified by \(\text{ChildSpec}\).

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a SelectionBox widget</td>
</tr>
<tr>
<td>ChildSpec</td>
<td>Atom, possible values: apply_button, cancel_button, default_button, help_button, list, list_label, ok_button, selection_label, separator, text, text, work_area</td>
</tr>
<tr>
<td>Child</td>
<td>Atom, the name of a widget</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

OSF/Motif XmSelectionBoxGetChild()
Test protocol facilities

**xmSendClientMessage** ( +WidgetName, +WindowId, +Property, +Protocol, +Message )

xmSendClientMessage/5 sends a message *Message* to the owner of the window *WindowId*. This window must be known to the display specified by the *WidgetName*. The message will be sent via the *Protocol* registered in *Property*. A receiver can request information about the arrival of a message by means of the appropriate protocol callback.

If the receiver is a IF/Prolog system, it can obtain the message by accessing the event component eventDataMsg.

A C receiver will obtain an XClientMessageEvent, in which the event structure is used as follows:

- `message-type`: (Atom) Property
- `format`: 32
- `data.l[0]`: (Atom) Protocol
- `data.l[1]`: (Time) timestamp

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **WindowId**: Integer, the window id
- **Property**: Atom
- **Protocol**: Atom
- **Message**: Atom

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xmAddProtocolCallback/5
- X: XSendEvent()
xmSetCallbackComponents/1 can be used to set callback components in the callback structure of the current callback. This makes only sense for in/out members of callback structures which have been introduced for Drag & Drop callbacks. Please consult the OSF/Motif reference manuals for further details.

**Arguments**

ArgList  
A list of value assignments for resources.  
These value assignments have the form: [ResourceName = ResourceValue, ...] or [ResourceName(ResourceValue), ...]

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **system_error(no_callback_struct_to_analyze)**  
The predicate xmGetCallbackComponent(s)/1 may only be called from within a callback.

- **domain_error(component_list)**  
The elements specified in the list must be value assignments.

- **existence_error(callback_struct_to_analyse)**  
There exists no callback structure that could be analysed.

**Compatibility**

xmSetCallbackComponents/1 is not available in the interface to OSF/Motif 1.1.

**See also**

OSF/Motif  
XmSetCallbackComponents()
Define the font size

Predicates

\texttt{xmSetFontUnit} ( +Unit )

\texttt{xmSetFontUnits} ( +HorizUnit, +VertUnit )

These predicates affect the sizes of the fonts used in widgets from the classes Gadget, Manager, and Primitive. This is useful when an application needs to specify a font size which is independent of the resolution of the screen. This applies if the unitType resource for the widgets is set to the value '100th_font_units' (see also the predicate \texttt{xmConvertUnits/6}).

\texttt{xmSetFontUnit/1} sets the vertical and horizontal font sizes to the value \textit{Unit}.

\texttt{xmSetFontUnits/2} allows the vertical and horizontal sizes of the font to be set separately.

Arguments

\begin{tabular}{ll}
  Unit & Integer \\
  HorizUnit & Integer \\
  VertUnit & Integer \\
\end{tabular}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page \pageref{exceptions}).

See also

\begin{itemize}
  \item IF/Prolog \hspace{1cm} \texttt{xmConvertUnits/6}
  \item OSF/Motif \hspace{1cm} \texttt{XmSetFontUnit()},...s()
\end{itemize}
xmSetMenuCursor/1

Set menu pointer

\[
\text{xmSetMenuCursor} \ (\ +\text{Cursor} )
\]

xmSetMenuCursor/1 sets the form of the mouse pointer for menus to \textit{Cursor}.
The pointer is assigned to the menu system and will be output whenever the application posts a menu.

**Arguments**

- \textit{Cursor} \quad \text{Atom, a cursor name}
  
  \text{Examples: arrow, mouse, pencil}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \quad \text{xmGetMenuCursor/1}
- OSF/Motif \quad \text{XmSetMenuCursor()}

Predicates

**xmTargetsAreCompatible/2**

Test whether targets of drop site and source are compatible

| **xmTargetsAreCompatible** ( +ExportTargetList, +ImportTargetList ) |

xmTargetsAreCompatible/2 tests whether given lists of export targets $ExportTargetListList$ and import targets $ImportTargetList$ match; i.e. contain at least one common element.

**Arguments**

- **ExportTargetList** List of atoms
- **ImportTargetList** List of atoms

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmTargetsAreCompatible/2 is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif XmTargetsAreCompatible()
xmTextClearSelection/1

Cancel first marking in the Text widget

```
xmTextClearSelection ( +TextWidget )
```

xmTextClearSelection/1 cancels the first marking in a text area in the TextWidget widget.

**Arguments**

- TextWidget: Atom, the name of a Text widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xmTextGetSelection/2, ...SetSelection/3
- OSF/Motif: XmTextClearSelection()
Copy text into the Text widget’s clipboard

\[
\text{xmTextCopy} \ ( \ +\text{TextWidget} \ ) \\
\text{xmTextCut} \ ( \ +\text{TextWidget} \ )
\]

These predicates copy items of text marked in the \textit{TextWidget} widget into the clipboard. \text{xmTextCopy/1} copies the first marked text item into the clipboard. \text{xmTextCut/1} copies the first marked text item into the clipboard. The marked text item is then deleted from the text.

Arguments

\begin{itemize}
  \item TextWidget \ Atom, the name of a Text widget
\end{itemize}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \text{environment\_error(text\_function\_failed)}
  - There is no text in the clipboard

See also

- IF/Prolog \ \text{xmTextPaste/2}
- OSF/Motif \ \text{XmTextCopy(),...Cut()}

xmTextDisableRedisplay/1

Prevent visual update of Text widget

```
xmTextDisableRedisplay (+TextWidget)
```

xmTextDisableRedisplay/1 prevents visual update of a text widget `TextWidget`. Thus multiple changes can be made and then be displayed in one go.

**Arguments**

- `TextWidget` Atom, the name of a text widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmTextDisableRedisplay/1 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog `xmTextEnableRedisplay/1`
- OSF/Motif `XmTextDisableRedisplay()`
Force visual update of Text widget

```
xmTextEnableRedisplay ( +TextWidget )
```

xmTextEnableRedisplay/1 enables the immediate visual update of the text widget TextWidget which had been prevented by a prior call to xmTextDisableRedisplay/1.

**Arguments**

- **TextWidget** Atom, the name of a text widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmTextEnableRedisplay/1 is not available in the interface to OSF/Motif 1.1.

**See also**

- IF/Prolog      xmTextDisableRedisplay/1
- OSF/Motif      XmTextEnableRedisplay()
Predicates for processing a TextField widget

The predicates for the TextField widget correspond to those for the Text widget. The TextField predicates are listed below in alphabetical order. For further details, refer to the descriptions of the Text widget predicates with the corresponding names (xmText...).

xmTextFieldClearSelection( +TextFieldWidget )
xmTextFieldCopy( +TextFieldWidget )
xmTextFieldCut( +TextFieldWidget )
xmTextFieldFreeConstraints( +TextFieldWidget )
xmTextFieldGetBaseline( +TextFieldWidget, ?XPosition )
xmTextFieldGetEditable( +TextFieldWidget )
xmTextFieldGetInsertionPosition( +TextFieldWidget, ?TextPosition )
xmTextFieldGetLastPosition( +TextFieldWidget, ?TextPosition )
xmTextFieldGetMaxLength( +TextFieldWidget, ?MaxLength )
xmTextFieldGetSelection( +TextFieldWidget, ?Selection )
xmTextFieldGetString( +TextFieldWidget, ?String )
xmTextFieldGetString( +TextFieldWidget, +FromTextPosition, String )
xmTextFieldGetString( +TextFieldWidget, +FromTextPosition, +ToTextPosition, ?String )
xmTextFieldGetSubstring( +TextFieldWidget, +StartPosition, +NumberChars, ?Text )
xmTextFieldInsert( +TextWidget, +TextPosition, +String )
xmTextFieldPaste( +TextFieldWidget )
xmTextFieldPosToXY( +TextFieldWidget, +TextPosition, ?XPosition, ?YPosition )
xmTextFieldRemove( +TextFieldWidget )
xmTextFieldReplace( +TextFieldWidget, +FromTextPosition, +ToTextPosition, +String )
xmTextFieldSetAddMode( +TextFieldWidget, +Boolean )
xmTextFieldSetConstraints( +TextFieldWidget, +ModifyStart, +ModifyEnd, +CursorStart, +CursorEnd )
xmTextFieldSetEditable( +TextFieldWidget, +Boolean )
Predicates

xmTextFieldSetHighlight( +TextFieldWidget, +FromTextPosition, +ToTextPosition, +HighlightMode )
xmTextFieldSetInsertionPosition( +TextFieldWidget, +TextPosition )
xmTextFieldSetMaxLength( +TextFieldWidget, +MaxLength )
xmTextFieldSetSelection( +TextFieldWidget, +FromTextPosition, +ToTextPosition )
xmTextFieldSetString( +TextFieldWidget, +String )
xmTextFieldShowPosition( +TextFieldWidget, +TextPosition )
xmTextFieldXYToPos( +TextFieldWidget, +XPosition, +YPosition, TextPosition )
xmTextFreeConstraints/1 cancels restrictions which have been imposed on user inputs into the TextWidget widget. Restrictions can be imposed identifying an area in the TextWidget widget which the user cannot access by movements of the insertion cursor, and which area may be modified.

As there is a limit on the number of widgets for which constraints may be defined (currently ten), the restrictions should be canceled when the widget is destroyed.

**Arguments**

TextWidget Atom, the name of a widget of type Text

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

IF/Prolog xmTextGetConstraints/5, ...SetConstraints/5
Query position of the first baseline in a Text widget

xmTextGetBaseline/2

xmTextGetBaseline/2 returns the Y coordinate $YPosition$ of the first baseline of the $TextWidget$ widget, relative to the top border of the widget. A baseline is a line on which a line of text is output.

In calculating the position of the first baseline, the following information is taken into account:

- the distance between the top border of the widget and the text (the marginHeight resource)
- the width of the shadow
- the width of the highlighting
- the size of the first font in the font list.

Arguments

- **TextWidget**: Atom, the name of a Text widget
- **YPosition**: Integer

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- OSF/Motif: XmTextGetBaseline()
Query restrictions on user inputs into the Text widget

```
```

xmTextGetConstraints/5 returns details of the restrictions which have been set on the input of text by the user into the `TextWidget` widget.

The following restrictions can be set:

- the area which can be modified, from `ModifyStart` to `ModifyEnd`
- the area within the `TextWidget` widget, from `CursorStart` to `CursorEnd`, which is to be accessible by moving the insertion cursor.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextWidget</td>
<td>Atom, the name of a widget of type Text</td>
</tr>
<tr>
<td>ModifyStart</td>
<td>Integer ≥ 0</td>
</tr>
<tr>
<td>ModifyEnd</td>
<td>Integer ≥ 0</td>
</tr>
<tr>
<td>CursorStart</td>
<td>Integer ≥ 0</td>
</tr>
<tr>
<td>CursorEnd</td>
<td>Integer ≥ 0</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

IF/Prolog       xmTextFreeConstraints/1, ...SetConstraints/5,
Check that a text item in the Text widget may be edited

```
xmTextGetEditable ( +TextWidget )
```

xmTextGetEditable/1 checks the value of the `editable` resource for the `TextWidget` widget. If the value is `true`, the goal succeeds, i.e. the text can be edited. If the value is `false`, the goal fails, i.e. the text cannot be edited.

**Arguments**

- `TextWidget` Atom, the name of a widget of type Text

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog `xmSetEditable/2`
- OSF/Motif `XmGetEditable()`
xmTextGetInsertionPosition/2 returns the position \textit{TextPosition} of the insertion cursor within a text item in the \textit{TextWidget} widget. This is the value of the cursorPosition resource. The position is determined by counting the number of characters from the start of the text buffer. The position of the first character is 0.

**Arguments**

- TextWidget: Atom, the name of a widget of type Text
- TextPosition: Integer $\geq 0$

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: \texttt{xmTextSetInsertionPosition/2}
- OSF/Motif: \texttt{XmTextGetInsertionPosition()}
Query position of last character in the Text widget

\[
\text{xmTextGetLastPosition} \ (\ +\text{TextWidget}, \ ?\text{TextPosition})
\]

xmTextGetLastPosition/2 returns the position \text{TextPosition} of the last character in the text buffer of the \text{TextWidget} widget. The position is determined by counting the number of characters from the start of the text buffer. The position of the first character is 0.

**Arguments**

- \text{TextWidget} \quad \text{Atom, the name of a widget of type Text}
- \text{TextPosition} \quad \text{Integer} \geq 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif \quad \text{XmTextGetLastPosition()}
xmTextGetMaxLength/2

Query the maximum length of text in the Text widget

```
xmTextGetMaxLength ( +TextWidget, ?MaxLength )
```

xmTextGetMaxLength/2 returns MaxLength, which is the maximum length of text which may be input to the TextWidget widget. This is the value of the maxLength resource.

**Arguments**

- TextWidget: Atom, the name of a widget of type Text
- MaxLength: Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmTextSetMaxLength/2`
- OSF/Motif: `XmTextGetMaxLength()`
Query marked text in the Text widget

\[
\text{xmTextGetSelection} ( \ +\text{TextWidget}, \ ?\text{Selection} )
\]

xmTextGetSelection/2 returns the first marked text item in the \textit{TextWidget} widget.

**Arguments**

- \texttt{TextWidget} Atom, the name of a widget of type Text
- \texttt{Selection} A list of atoms (lines)

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \texttt{xmTextSetSelection/3}
- OSF/Motif \texttt{XmTextGetSelection()}

Motif Manual 354 IF/Prolog V5.2
Query the position of marked text in the Text widget

```
```

xmTextGetSelectionPosition/3 returns the positions of the left and right bounds of the first marked character string in the text buffer of the TextWidget widget. LeftTextPosition is the position of the left-hand limit, i.e. the start of the marked character string. RightTextPosition is the position of the right-hand limit, i.e. the end of the character string. The positions are determined by counting the number of characters from the start of the text buffer. The position of the first character in the text buffer is 0.

**Arguments**

- **TextWidget**: Atom, the name of a widget of type Text
- **LeftTextPosition**: Integer ≥ 0
- **RightTextPosition**: Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmTextGetSelectionPosition()
Query character string in the Text widget

xmTextGetString ( +TextWidget, ?String )

xmTextGetString ( +TextWidget, +FromTextPosition, ?String )

xmTextGetString ( +TextWidget, +FromTextPosition, +ToTextPosition, ?String )

These predicates supply the character string String from the text buffer of the TextWidget widget.

To avoid having to process the complete character string every time, there are three formats for this predicate. The complete character string can be accessed (first format). The character string can be accessed starting from a particular position, FromTextPosition (second format). Or the character string within a defined range of text can be accessed, from FromTextPosition to ToTextPosition (third format). The position of the first character in the text buffer is 0.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextWidget</td>
<td>Atom, the name of a widget of type Text</td>
</tr>
<tr>
<td>String</td>
<td>A list of atoms (lines)</td>
</tr>
<tr>
<td>FromTextPosition</td>
<td>Integer ≥ 0</td>
</tr>
<tr>
<td>ToTextPosition</td>
<td>Integer ≥ 0</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

<table>
<thead>
<tr>
<th>System</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xmTextSetString/2</td>
</tr>
<tr>
<td>OSF/Motif</td>
<td>XmTextGetString()</td>
</tr>
</tbody>
</table>
Retrieve a portion of the text in a text widget

xmTextGetSubstring ( +TextWidget, +StartPosition, +NumberChars, ?Text )

xmTextGetSubstring/4 retrieves a substring of a text widget TextWidget starting at position StartPosition and of length NumberChars.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextWidget</td>
<td>Atom, the name of a text widget</td>
</tr>
<tr>
<td>StartPosition</td>
<td>Integer ≥ 0</td>
</tr>
<tr>
<td>NumberChars</td>
<td>Integer ≥ 0</td>
</tr>
<tr>
<td>Text</td>
<td>List of lines</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

xmTextGetSubstring/4 is not available in the interface to OSF/Motif 1.1.

See also

<table>
<thead>
<tr>
<th>Language</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xmTextGetSubstring/4</td>
<td>xmTextGetString/4</td>
</tr>
<tr>
<td>OSF/Motif</td>
<td>XmTextGetSubstring()</td>
<td></td>
</tr>
</tbody>
</table>
Query position of the first character in the Text widget

\[ \text{xmTextGetTopCharacter} \left( +\text{TextWidget}, \ ?\text{TopTextPosition} \right) \]

\text{xmTextGetTopCharacter/2} returns the position \textit{TopTextPosition} of the first visible character which will be output in the \textit{TextWidget} widget. The position is determined by counting the number of characters from the start of the text buffer. The position of the first character in the text buffer is 0.

**Arguments**

- \text{TextWidget} \quad \text{Atom, the name of a widget of type Text}
- \text{TopTextPosition} \quad \text{Integer} \geq 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \quad \text{xmTextSetTopCharacter/2}
- OSF/Motif \quad \text{XmTextGetTopCharacter()}

xmTextInsert/3

Insert text into the Text widget

\[
\text{xmTextInsert} \ ( +\text{TextWidget}, +\text{TextPosition}, +\text{String} )
\]

xmTextInsert/3 inserts the character string \textit{String} into the text buffer of the \textit{TextWidget} widget, at the position given by \textit{TextPosition}. The position of the first character in the text buffer is 0.

**Arguments**

- \textit{TextWidget} \quad Atom, the name of a widget of type Text
- \textit{TextPosition} \quad Integer $\geq$ 0
- \textit{String} \quad Atom, or a list of atoms (lines)

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif \quad XmTextInsert()
Predicates \hspace{10cm} \texttt{xmTextPaste/1}

**Insert text from the Text widget clipboard**

\[
\texttt{xmTextPaste} \ ( \ +\text{TextWidget} \ )
\]

\texttt{xmTextPaste/1} inserts the contents of the clipboard into the text area of the \textit{TextWidget} widget at the position of the insertion cursor. The predicates \texttt{xmTextCut/1} and \texttt{xmTextCopy/1} copy text into the clipboard.

The value of the pendingDelete resource for the \textit{TextWidget} widget determines whether the marked text, at the place where the new text is being inserted, will be deleted or not.

**Arguments**

- \texttt{TextWidget} Atom, the name of a widget of type Text

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\[
\texttt{environment\_error(text\_function\_failed)}
\]

There is no text in the clipboard

**See also**

- IF/Prolog \texttt{xmTextCopy/1, \ldots Cut/1,}
- OSF/Motif \texttt{XmTextPaste()}

Motif Manual 360 IF/Prolog V5.2
Query the position of a character in the Text widget

\texttt{xmTextPosToXY} ( \texttt{+TextWidget}, \texttt{+TextPosition}, \texttt{?XPosition}, \texttt{?YPosition} )

\texttt{xmTextPosToXY} returns the X and Y coordinates, \texttt{XPosition} and \texttt{YPosition}, of the character which is stored in the \textit{TextWidget} widget at the position defined by \texttt{TextPosition}. These X and Y coordinates are relative to the top left corner of the \textit{TextWidget} widget.

The position of the first character in the text buffer is 0.

\textbf{Arguments}

- \texttt{TextWidget} Atom, the name of a Text widget
- \texttt{TextPosition} Integer \( \geq 0 \)
- \texttt{XPosition} Integer
- \texttt{YPosition} Integer

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

- IF/Prolog \texttt{xmTextXYToPos/4}
- OSF/Motif \texttt{XmTextPosToXY()}
Delete text from the Text widget

\[
\textit{xmTextRemove} ( \ +\text{TextWidget} )
\]

\textit{xmTextRemove/1} deletes the first marked item of text in the \textit{TextWidget} widget.

**Arguments**

TextWidget Atom, the name of a Text widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \textit{xmTextInsert/3}, \textit{....Replace/4}
- OSF/Motif \textit{XmTextRemove()}
xmTextReplace/4

Replace text in the Text widget

```
xmTextReplace ( +TextWidget, +FromTextPosition, +ToTextPosition, +String )
```

xmTextReplace/4 replaces the text area between the positions `FromTextPosition` and `ToTextPosition` in the `TextWidget` widget, by the character string `String`.

The position of the first character in the text buffer is 0.

**Arguments**

- **TextWidget**: Atom, the name of a Text widget
- **FromTextPosition**: Integer ≥ 0
- **ToTextPosition**: Integer ≥ 0
- **String**: Atom or list of atoms (lines)

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: `xmTextInsert/3`, `...Remove/1`
- OSF/Motif: `XmTextReplace()`
Predicates

**xmTextScroll/2**

**Move text in the Text widget**

```prolog
xmTextScroll ( +TextWidget, +Lines )
```

`xmTextScroll/2` scrolls the area of text which is visible in the `TextWidget` widget by the specified number of lines. A positive value for `Lines` moves the text upwards, a negative one moves it downwards.

**Arguments**

- `TextWidget`: Atom, the name of a Text widget
- `Lines`: Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif: `XmTextScroll()`
Set add mode for insertion of text into the Text widget

```
xmTextSetAddMode ( +TextWidget, +Mode )
```

xmTextSetAddMode/2 sets the mode which determines whether or not it is possible to select text in a widget TextWidget by means of the keyboard.

If Mode has the value true, then it is possible to mark a number of entries in the list with one key depression. If Mode has the value false, then only one entry can be marked. If another entry is then marked, the previous marking will be canceled.

**Arguments**

- **TextWidget** Atom, the name of a Text widget
- **Mode** Atom: true | false | yes | no | on | off

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif XmTextSetAddMode()
Specify restrictions on user inputs into a Text widget

\[
\text{xmTextSetConstraints} ( +\text{TextWidget}, ?\text{ModifyStart}, ?\text{ModifyEnd}, ?\text{CursorStart}, ?\text{CursorEnd} )
\]

xmTextSetConstraints/5 sets restrictions on user inputs into the \textit{TextWidget} widget.

The following restrictions can be set:

- the area of the \textit{TextWidget} widget which can be modified, from \textit{ModifyStart} to \textit{ModifyEnd}
- the area within the \textit{TextWidget} widget, from \textit{CursorStart} to \textit{CursorEnd}, which is to be accessible to the insertion cursor.

The restrictions only take effect when the appropriate callback is activated; i.e. is at least declared by a dummy predicate. If a variable is specified for any of the arguments, the value previously registered remains in effect.

The default values are:

- \textit{ModifyStart} 0,
- \textit{ModifyEnd} maxint,
- \textit{CursorStart} 0,
- \textit{CursorEnd} maxint.

Arguments

- \textit{TextWidget} Atom, the name of a Text widget
- \textit{ModifyStart} Variable or integer \( \geq 0 \)
- \textit{ModifyEnd} Variable or integer \( \geq 0 \)
- \textit{CursorStart} Variable or integer \( \geq 0 \)
- \textit{CursorEnd} Variable or integer \( \geq 0 \)

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog \textit{xmTextFreeConstraints/1, ...GetConstraints/5}
Specify whether a text item in the Text widget can be edited

```
xmTextSetEditable ( +TextWidget, +Editable )
```

`xmTextSetEditable/2` sets the value of the `editable` resource for the `TextWidget` widget. If the value of `Editable` is `true`, the text may be edited; if its value is `false`, the text cannot be edited.

**Arguments**

- `TextWidget` Atom, the name of a Text widget
- `Editable` Atom: `true` | `false` | `yes` | `no` | `on` | `off`

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog `xmTextGetEditable/1`
- OSF/Motif `XmTextSetEditable()`
Set highlighting for a text item in the Text widget

\[
\text{XMTextSetHighlight} \ ( \ +\text{TextWidget}, \ +\text{FromTextPosition}, \ +\text{ToTextPosition}, \\
\quad +\text{HighlightMode} )
\]

xmTextSetHighlight/4 sets the highlighting of a text area in the TextWidget widget.

This highlighting starts with the character in the position FromTextPosition and runs to the character in the position ToTextPosition. The position of the first character in the text buffer is 0.

HighlightMode defines how the highlighting is to be displayed.

Arguments

- TextWidget: Atom, the name of a Text widget
- FromTextPosition: Integer \( \geq 0 \)
- ToTextPosition: Integer \( \geq 0 \)
- HighlightMode: Atom: highlight_normal | highlight_selected | highlight_secondary_selected

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- OSF/Motif: XmSetHighlight()
Set the position of the insertion cursor in the Text widget

```
xmTextSetInsertionPosition(+TextWidget, +TextPosition)
```

`xmTextSetInsertionPosition/2` sets the position `TextPosition` of the insertion cursor within a text item in the `TextWidget` widget. This is the value of the `cursorPosition` resource. The position is determined by counting the number of characters from the start of the text buffer. The position of the first character is 0.

**Arguments**

- **TextWidget** Atom, the name of a Text widget
- **TextPosition** Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog `xmGetInsertionPosition/2`
- OSF/Motif `XmTextSetInsertionPosition()`
Set the maximum length of character string in the Text widget

```
xmTextSetMaxLength ( +TextWidget, +MaxLength )
```

xmTextSetMaxLength/2 sets the length of the longest character string that the user may input into the TextWidget widget. This is the value of the maxLength resource.

**Arguments**

- **TextWidget**  
  Atom, the name of a Text widget
- **MaxLength**  
  Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog  
  `xmTextGetMaxLength/2`
- OSF/Motif  
  `XmTextSetMaxLength()`
xmTextSetSelection/3 sets the first text marking in the Text widget. In addition, this predicate sets the insertion cursor to the last position in the marked area of text. The marking is set between the positions defined by FromTextPosition and ToTextPosition. The position of the first character in the text buffer is 0.

**Arguments**

- **TextWidget**: Atom, the name of a Text widget
- **FromTextPosition**: Integer ≥ 0
- **ToTextPosition**: Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xmTextGetSelection/2
- OSF/Motif: XmTextSetSelection()
Predicates  

**Specify common sources for Text widgets**

| xmTextSetSource ( +DestTextWidget, +SrcTextWidget, +TopTextPosition, +CursorPosition ) |

Widgets of the type Text can access common sources of text. `xmTextSetSource/4` specifies that the widget named `DestTextWidget` accesses the text source of the widget identified by `SrcTextWidget`.

Thereafter both widgets display the same text, though possibly a different section thereof. The section in the `DestTextWidget` widget can be specified using the parameter `TopTextPosition`, and the cursor position by means of `CursorPosition`.

Both widgets then have equal rights to modify the section of text. The text in the `DestTextWidget` will be retained, even if the source for `SrcTextWidget` is later changed. This behavior can be used, in particular, to preserve a text source for later use. To achieve this, an unmanaged Text widget can be created, and the source which is temporarily not required can be assigned to this.

**Arguments**

- **DestTextWidget**: Atom, the name of a Text widget
- **SrcTextWidget**: Atom, the name of a Text widget
- **TopTextPosition**: Integer ≥ 0
- **CursorPosition**: Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif  
  XmTextSetSource()
Specify contents of a Text widget

\texttt{xmTextSetString} ( +TextWidget, +String )

\texttt{xmTextSetString/2} defines \textit{String}, which is a text item to be displayed in the \textit{TextWidget} widget. In addition, this predicate sets the insertion cursor to the start of the text item.

**Arguments**

- \texttt{TextWidget} \hspace{1em} Atom, the name of a Text widget
- \texttt{String} \hspace{1em} Atom, or a list of atoms (lines)

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \hspace{1em} \texttt{xmTextGetString/2}
- OSF/Motif \hspace{1em} \texttt{XmTextSetString()}
Set the position of the first character in the Text widget

\[
\text{xmTextSetTopCharacter} \ ( \ +\text{TextWidget}, \ +\text{TopTextPosition} \ )
\]

xmTextSetTopCharacter/2 sets TopTextPosition, the position of the first character displayed in the TextWidget widget. The position is determined by counting the number of characters from the start of the text buffer. The position of the first character in the text buffer is 0.

**Arguments**

- TextWidget Atom, the name of a Text widget
- TopTextPosition Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \texttt{xmTextGetTopCharacter/2}
- OSF/Motif XmTextSetTopCharacter()
xmTextShowPosition/2 ensures that the position in the Text Widget widget defined by Text-Position is visible. The position is determined by counting the number of characters from the start of the text buffer. The position of the first character in the text buffer is 0.

**Arguments**

- **TextWidget**: Atom, the name of a Text widget
- **TextPosition**: Integer ≥ 0

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif: XmTextShowPosition()
Query the position of a character in the Text widget

xmTextXYToPos ( +TextWidget, +XPosition, +YPosition, ?TextPosition )

xmTextXYToPos/4 returns the position TextPosition of the character in the TextWidget widget which is located nearest to the coordinates XPosition and YPosition. These X and Y coordinates are relative to the top left corner of the Text widget.

The position of the first character in the text buffer is 0.

Arguments

- TextWidget: Atom, the name of a Text widget
- XPosition: Integer
- YPosition: Integer
- TextPosition: Integer ≥ 0

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog: xmTextPosToXY/4
- OSF/Motif: XmTextXYToPos()
Check and set status of the ToggleButton gadget

\begin{align*}
\text{xmToggleButtonGadgetGetState} & ( +\text{ToggleGadget} ) \\
\text{xmToggleButtonGadgetSetState} & ( +\text{ToggleGadget}, +\text{State}, +\text{Notify} )
\end{align*}

These predicates manipulate the status of the ToggleButton gadget.

\text{xmToggleButtonGadgetGetState/1} returns the status of the \textit{ToggleButton} gadget. The goal will succeed if the gadget is marked, and will fail if it is not marked.

\text{xmToggleButtonGadgetSetState/3} sets the status \textit{State} of the \textit{ToggleButton} gadget. If the value of \textit{State} is \texttt{true}, the gadget will be marked; if it is \texttt{false}, the marking will be canceled. \textit{Notify} specifies whether the callback \texttt{valueChangedCallback} is called or not.

\textbf{Arguments}

\begin{align*}
\text{ToggleGadget} & \quad \text{Atom, the name of a ToggleButton gadget} \\
\text{State} & \quad \text{Atom: true | false | yes | no | on | off} \\
\text{Notify} & \quad \text{Atom: true | false | yes | no | on | off}
\end{align*}

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

\begin{align*}
\text{OSF/Motif} & \quad \text{XmToggleButtonGadgetSetState(), ...GetState()}
\end{align*}
Select a widget

xmTrackingEvent/4 serves to determine the widget where a button press occurred. This predicate can be used in the same way as xmTrackingLocate/4 and also works for widgets that do not handle their input events themselves.

Arguments

- **WidgetName**: Atom, the name of a widget
- **Cursor**: Atom, name of a cursor
- **ConfineTo**: Atom, the name of a widget
- **SelectedWidget**: Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

xmTrackingEvent/4 is not available in the interface to OSF/Motif 1.1.

See also

- IF/Prolog: xmTrackingLocate/4
- OSF/Motif: XmTrackingEvent()
xmTrackingLocate/4

Select a widget

xmTrackingLocate ( +WidgetName, +Cursor, +ConfineTo, ?SelectedWidget )

xmTrackingLocate/4 provides a modal interface for selection of a component. It is intended to support context help. The function grabs the pointer and returns the widget in which a button press occurred.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Cursor</td>
<td>Atom, a cursor name</td>
</tr>
<tr>
<td></td>
<td>Examples: arrow, mouse, pencil</td>
</tr>
<tr>
<td>ConfineTo</td>
<td>Atom, the name of a widget or none</td>
</tr>
<tr>
<td>SelectedWidget</td>
<td>Atom, the name of a widget</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

OSF/Motif      XmTrackingLocate()
Initiate execution of expose events

\[
\text{xmUpdateDisplay} \ ( \ +\text{WidgetName} \ )
\]

xmUpdateDisplay/1 requests that all pending expose events are immediately executed and deleted from the queue. The widget named \textit{WidgetName} may be any widget of the application.

**Arguments**

- \textit{WidgetName} \quad Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- OSF/Motif \quad \texttt{XmUpdateDisplay()}

Query the version of OSF/Motif

`xmVersion ( ?Version, ?Revision )`

`xmVersion/2` returns the version number `Version` of the OSF/Motif version and the release number `Revision` of the OSF/Motif library which are currently being used.

**Arguments**

- **Version**
  - Integer, version of OSF/Motif
- **Revision**
  - Integer, the release number of OSF/Motif
Retrieve Baseline information for a widget

\[
\text{xmWidgetGetBaselines} \ ( \ +\text{WidgetName}, \ ?\text{BaseLines} )
\]

xmWidgetGetBaselines/2 returns a list of baselines of widget \text{WidgetName} in the list \text{BaseLines}.

**Arguments**

- \text{WidgetName} : Atom, the name of a widget
- \text{BaseLines} : List of integers

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xmWidgetGetBaselines/2 is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif : XmWidgetGetBaselines()
**Retrieve Display Rectangle information for a widget**

`xmWidgetGetDisplayRect( +WidgetName, ?DisplayRectangle )`

`xmWidgetGetDisplayRect/2` returns the x and y-coordinates and the width and height of the display rectangle of widget `WidgetName` in the list `DisplayRectangle`.

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **DisplayRectangle**: List, [x,y,width,height]

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

`xmWidgetGetDisplayRect/2` is not available in the interface to OSF/Motif 1.1.

**See also**

- OSF/Motif: `XmWidgetGetDisplayRect()`
Add an action predicate

xtAddAction ( +ActionName )

xtAddActions ( +ActionList )

These predicates manipulate the list of action predicates. A maximum of ten global actions can be defined. There are two predefined global actions, these being: xtActionBreak, to exit from the main loop of the program, and xtActionBye, to exit from IF/Prolog.

xtAddAction/1 adds the action predicate specified by ActionName to those which have already been declared and which can be used when specifying translations for widgets.

xtAddAction/1 checks whether the action predicate ActionName already exists, and results in an exception if ActionName is not found.

xtAddActions/1 calls xtAddAction/1 to add each of the action predicates in the ActionList.

The action predicate will be called if an event defined in a translation table is linked in the table to the action, and if the event then occurs. The format of the action predicate call is as follows:

Predicate( Widget, Params ).

The meaning of the arguments is as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widget</td>
<td>Atom, the name of the widget for which the action has been initiated</td>
</tr>
<tr>
<td>Params</td>
<td>A structure with an arity of n and the functor params, which contains the arguments of the action specified in the translation table in the form of atoms. If no arguments were used there, the only information passed will be the atom params.</td>
</tr>
</tbody>
</table>

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionName</td>
<td>Atom The functor of an action predicate with an arity of two</td>
</tr>
<tr>
<td>ActionList</td>
<td>A list of atoms These atoms are the functors of action predicates with an arity of two</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

representation_error(exceeded_max_actions)

The maximum number of global actions which can be defined is 10
**Example**

bsp1 :-
    xtInitialize( toplevel, toplevel, 'Action', [], [action]),
    xmCreateLabel( label, toplevel, label, [ labelString = label]),
    xtAddAction(actionDel),
    xtSetValue(label, translations =
        '#augment
<Key> D: actionDel( ) 
<Key> End: xtActionBye( )'),
    xtManageChild( label),
    xtRealizeWidget( toplevel),
    xtMainLoop.

    actionDel( W, _) :-
        xtDestroyWidget( W),
        write( actiondestroy),
        nl,
        xtBreakMainLoop.

**See also**

X  
XtAddAction(), ...s()
Add callback predicate to callback list

| xtAddCallback (  +WidgetName, +CallbackName, +CallbackPredicate,  
| +ClientData ) |

| xtAddCallbacks (  +CallbackList ) |

These predicates add callback predicates to a callback list.

xtAddCallback/4 adds the callback predicate CallbackPredicate to the callback list CallbackName for the widget named WidgetName. In doing so, xtAddCallback/4 checks whether the callback predicate CallbackPredicate exists, and initiates an exception if it cannot find the predicate.

ClientData passes application-specific data to the predicate.

xtAddCallbacks/1 calls xtAddCallback/4 for each of the predicates identified in the CallbackList.

The callback predicate will be called up if a widget activates the corresponding callback. The format of the call is as follows:

Predicate( Widget, ClientData, Reason )

The meaning of the arguments is as follows:

- Widget: Atom, the name of the widget which initiates the callback.
- ClientData: Term, as defined in the associated predicate xtAddCallback/4.
- Reason: Atom, details of why the callback was initiated (cf. section entitled Setting and modifying resources on page 23).

Additional data about a callback which has been initiated can, if necessary, be obtained using the predicates xmGetCallbackComponent(s)/1 or xtGetEventComponent(s)/1, as appropriate.

The following callback predicates are predefined and can be used for pop-up shells:

- xtCallbackPopdown/3: The pop-up shell Shell defined by ClientData in the form [Shell, Enable] is terminated (pop-down), and it is then possible to activate the Enable widget again. This latter should be the widget which has the associated callback predicate, xtCallbackNone, xtCallbackExclusive or xtCallbackNonExclusive.
The following additional callback predicates are predefined:

- `xtCallbackBreak/3`: Calls `xtBreakMainLoop/0`.
- `xtCallbackBye/3`: Terminates IF/Prolog.
- `xtCallbackSetGlobal/3`: Calls `set_global(Var, Val)` if `ClientData` specifies `[Var, Val]`.

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

### Arguments

- **WidgetName**: Atom, the name of a widget
- **CallbackName**: Atom, the name of a callback
- **CallbackPredicate**: Atom, the functor of a predicate with an arity of three
- **ClientData**: Term
- **CallbackList**: List of lists These contain the following four elements: `[WidgetName, CallbackName, CallbackPredicate, ClientData]`.

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(callback_name)**
  - There is no callback list with the specified name
- **domain_error(widget_callback)**
  - The specified widget does not have the specified callback list.
- **system_error(callback_predicate_failed)**
  - A callback predicate may not fail.

### See also

- `XtAddCallback()`, `...()`, `XtCallbackNone()`, `XtCallbackExclusive()`, `XtCallbackNonExclusive()`, `XtCallbackPopdown()`, `XtCreatePopupShell()`
Add support for editres client

xtAddEditresSupport/1

xtAddEditresSupport/1 adds support for the editres client. This client lets you view the widget tree of applications and experimentally change resource values for widgets within the application. The widget hierarchy of ShellWidget will be used.

Arguments

ShellWidget Atom, the name of a shell widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Example

... xtAppCreateShell(Kid_shell, Kid_shell, prolog_help, applicationShellWidgetClass, []), xtAddEditresSupport(Kid_shell), ...

Compatibility

xtAddEditresSupport/1 is not available in the interface to OSF/Motif 1.1. xtAddEditresSupport/1 is not available on some platforms.

See also

XtAddEditresSupport()
xtAddGrab/3

Restrict input to a particular widget

\[
\text{xtAddGrab} \left( \text{+WidgetItem, +Exclusive, +SpringLoaded} \right)
\]

xtAddGrab/3 inserts the widget \textit{WidgetItem} into a modifiable widget cascade. This defines the sequence in which user events will be passed on. Examples of modifiable widget cascades are menus or dialog boxes, for which a mouse click is required to output the entire widget cascade (pop-ups).

xtAddGrab/3 restricts a user’s input to a particular widget \textit{WidgetItem}. \textit{Exclusive} specifies whether events which the user initiates will be passed exclusively to this widget, or will be passed in addition to the preceding widgets in the widget sequence. \textit{SpringLoaded} indicates whether the widget has been displayed because the user pressed a button.

If the 3rd argument of the predicate xtAddGrab/3 is \texttt{true}, the predicate checks if the 2nd argument is \texttt{true}. If not, a warning will be output.

Arguments

- **WidgetItem**: Atom, the name of a widget
- **Exclusive**: Atom: true | false | yes | no | on | off
- **SpringLoaded**: Atom: true | false | yes | no | on | off

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog \texttt{xtRemoveGrab/1}
- X \texttt{XtAddGrab()}
Set input predicate

xtAddInput/5 specifies that the input predicate \textit{InputPredicate} is to be called if an event specified in \textit{InputMask} occurs for the I/O medium specified by \textit{Source}.

Apart from standard streams or aliases a file or pipe opened by IF/Prolog may also be specified as the \textit{Source}. The name of widget may be not used at this point.

\textit{InputMask} must be xtInputReadMask, xtInputWriteMask or xtInputExceptMask.

\begin{itemize}
  \item \texttt{xtInputReadMask} \quad The input predicate will be activated if data can be read from the specified \textit{Source} medium without causing a block.
  \item \texttt{xtInputWriteMask} \quad The input predicate will be activated if data can be written to the specified \textit{Source} medium without causing a block. Outputs can cause a block by, for example, writing to a pipe when the data is not being read from its other end.
  \item \texttt{xtInputExceptMask} \quad The input predicate will be activated if an exception situation arises for the specified medium. In particular, this will be the case if EOF is reached.
\end{itemize}

\textit{InputPredicate} must be a predicate with two parameters. It will be called if a corresponding event is reported:

\texttt{InputPredicate( InputName, ClientData ).}

\textit{InputName} is a unique identifier assigned by the application to the input job declared by xtAddInput/5. This will be used by the input predicate and by xtRemoveInput/1 to identify the input job.

\textit{ClientData} passes application-specific data to the predicate.

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Caution
If an input event with the condition `xtInputReadMask` results in an input predicate being called, then the input predicate must read from the file with the specified file identifier `Source`, to ensure that the reason for the event is canceled again. Otherwise an endless loop will result.

Using `get/1` creates another problem. Due to operating system buffering, the input will be read from `Source`, but the callback will not be called again to permit the application to read the rest of the line. Consequently, it should be read line by line, using `get_until/3`.

To completely switch off the system buffering, the IF/Prolog system may also be called with the `-allflush` option.

`InputPredicate` should not fail. However, if it does, this will result in the exception `input_predicate_failed`.

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputName</td>
<td>Atom, job name</td>
</tr>
<tr>
<td>Source</td>
<td>Atom: stdin</td>
</tr>
<tr>
<td>InputMask</td>
<td>Atom: <code>xtInputReadMask</code></td>
</tr>
<tr>
<td>InputPredicate</td>
<td>Atom: the functor of a predicate with an arity of two</td>
</tr>
<tr>
<td>ClientData</td>
<td>Term</td>
</tr>
</tbody>
</table>

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **permission_error(create,input_predicate)**
  - There is already an input with the specified name.

- **system_error(input_predicate_failed)**
  - An input predicate which has been activated has not terminated successfully. (The exception is not raised until the input predicate is executed).

### See also

- IF/Prolog: `xtRemoveInput/1`
- X: `XtAddInput()`
Set timeout predicate

xtAddTimeOut ( +TimeOutName, +Interval, +TimeOutPredicate, +ClientData )

xtAddTimeOut/4 specifies that the predicate TimeOutPredicate is to be called after the time interval in milliseconds specified by Interval has elapsed.

TimeOutPredicate must be a predicate with two parameters. It will be called when a timeout event is reported:

TimeOutPredicate( TimeOutName, ClientData ).

TimeOutName is a unique identifier assigned by the application to the timeout job declared by xtAddTimeOut/4. It is used by the timeout predicate and by xtRemoveTimeOut/1 to identify the timeout job.

ClientData passes application-specific data to the predicate.

If TimeOutPredicate is called, the timeout request is deleted. If it is to be called again after a specified interval of time, then the timeout request must be reset in the timeout predicate.

Caution

TimeOutPredicate should not fail. However, if it does, this will result in the exception timeout_predicate_failed.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOutName</td>
<td>Atom, the job name</td>
</tr>
<tr>
<td>Interval</td>
<td>Integer</td>
</tr>
<tr>
<td>TimeOutPredicate</td>
<td>Atom, the functor of a predicate with an arity of two</td>
</tr>
<tr>
<td>ClientData</td>
<td>Term</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

permission_error(create, timeout_predicate)
There is already a timeout job with the name specified.

system_error(timeout_predicate_failed)
A timeout predicate which has been activated has not terminated successfully. (The exception is not raised until the timeout predicate is executed).
xtAddTimeOut/4  Predicates

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtRemoveTimeOut/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtAddTimeOut()</td>
</tr>
</tbody>
</table>
Set work predicate

```prolog
xtAddWorkProc ( +WorkName, +WorkPredicate, @ClientData )
```

xtAddWorkProc/3 specifies that the predicate \textit{WorkPredicate} is to be called when there are no events awaiting processing in the main loop of the program.

\textit{WorkPredicate} must be a predicate with two parameters. It will be called when there is no event waiting to be processed.

\textbf{WorkPredicate} ( WorkName, ClientData ).

\textit{WorkName} is a unique identifier assigned by the application to the work job declared by xtAddWorkProc/3. It is used by the work predicate and by xtRemoveWorkProc/1 to identify the work job.

\textit{ClientData} passes application-specific data to the predicate.

\textit{WorkPredicate} should only operate for fractions of a second, so that the user interface continues to appear active.

If \textit{WorkPredicate} fails the work job will be deleted.

**Arguments**

- **WorkName**: Atom, the job name
- **WorkPredicate**: Atom, the functor of a predicate with an arity of two
- **ClientData**: Term

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{work_name_already_used}

There is already a work job with the specified name.

**See also**

- IF/Prolog: xtRemoveWorkProc/1
- X: XtAddWorkProc()
xtAugmentTranslations/2

Augment Translations

\[ \text{xtAugmentTranslations} \left( \ +\text{WidgetName}, \ +\text{Translations} \right) \]

xtAugmentTranslations/2 is used to augment translations of a widget. If a translation of an event already exists this one will not be replaced. xtAugmentTranslations/2 is syntactically simpler than augmenting the translations with xtSetValue/2 (see the example). Translations are usually set in a resource file. You should only hardcode translations, if you don’t want them to be meddled with by a user.

Arguments

- **WidgetName** Atom, the name of a widget
- **Translations** Atom, event

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Example

The following calls have the same effect:

\[
\text{xtAugmentTranslations}( \text{widget}, \ 'Ctrl<Key>C: xtActionBye()' )
\]

\[
\text{xtSetValue}( \text{widget}, \n\quad [ \text{translations} = \n\quad \text{'#augment\n Ctrl<Key>C: xtActionBye()' } ] ).
\]

See also

- IF/Prolog xtOverrideTranslations/2, xtSetValue/2, xtSetValues/2
- X XtAugmentTranslations()
Predicates          xtBreakMainLoop/0

## Interrupt main loop

| xtBreakMainLoop |

xtBreakMainLoop/0 interrupts the processing of the main event loop in the program whenever events occur and callbacks have to be executed.

xtBreakMainLoop/0 is only effective if it is called from within a callback predicate, an action routine, or similar.

The interruption does not take effect until the currently active callback predicate terminates.

### See also

- IF/Prolog xtMainLoop/0
xtCallAcceptFocus/1

Determine readiness to accept keyboard focus

xtCallAcceptFocus( +WidgetName )

xtCallAcceptFocus/1 calls WidgetName’s accept_focus procedure and succeeds if the accept_focus procedure returns TRUE. xtCallAcceptFocus/1 does not actually set the keyboard focus but can be used to determine whether WidgetName would accept the keyboard focus if offered with xtSetKeyboardFocus/2.

Arguments

WidgetName Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtSetKeyboardFocus/2
X XtCallAcceptFocus()
Predicates xtCallActionProc/2

Call an action

xtCallActionProc ( +WidgetName, +Action )

xtCallActionProc/2 calls an action directly. Usually actions are only called indirectly via events which must have been bound to the action in a translation table.

Arguments

WidgetName Atom, the name of a widget
Action Atom, the name of an action

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtAddAction/2, xtAddActions/2
X XtCallActionProc()
Output class and name of a widget

\[
\texttt{# xtClass ( ?WidgetName, ?ClassName )}
\]

xtClass/2 returns the class \textit{ClassName} of the widget named \textit{WidgetName}.

If a variable is used for the argument \textit{WidgetName}, then for each widget which is known to IF/Prolog, its class will be unified with \textit{ClassName}. By backtracking, it will thus be possible to determine all pairings of widgets and widget classes.

**Arguments**

\begin{itemize}
  \item \texttt{WidgetName} \quad \text{Atom, the name of a widget}
  \item \texttt{ClassName} \quad \text{Atom, a defined class name}
\end{itemize}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

\begin{itemize}
  \item X \quad \text{XtClass()}
\end{itemize}
Predicates xtCreateApplicationShell/4

Create toplevel window

\[
\text{xtCreateApplicationShell ( ?WidgetName, +ToolkitName, +ClassName, +ArgList )}
\]

xtCreateApplicationShell/4 creates a toplevel widget \textit{WidgetName} with the name \textit{ToolkitName} as an additional toplevel window. This can be used in exactly the same way as the window created using xtInitialize/5.

\textit{WidgetName} defines the name under which the widget is known in IF/Prolog. If \textit{WidgetName} is a variable, then IF/Prolog assigns a unique name and unifies \textit{WidgetName} with this name.

\textit{ToolkitName} defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as \textit{WidgetName}.

Arguments

- \textit{WidgetName} \text{Atom}, the name of a widget
- \textit{ToolkitName} \text{Atom}
- \textit{ClassName} \text{Atom}, a defined Shell class name
- \textit{ArgList} \text{A list of value assignments for resources. These value assignments have the form: \{ResourceName = ResourceValue, \ldots\} or \{ResourceName(ResourceValue), \ldots\}}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \textbf{domain_error(unused\_widget\_name)}
  
  The specified widget name is already in use.

- \textbf{domain_error(internal\_class\_code)}
  
  The specified argument must be the name of a valid widget class.

Further exceptions result if \textit{ArgList} is incorrect. These are listed under the xtSetValues/2 predicate.

See also

- IF/Prolog \text{xtSetValues/2}
- X \text{XtCreateApplicationShell()}

Motif Manual 400 IF/Prolog V5.2
Create and manage child widget

xtCreateManagedWidget/5 creates a widget with the name \textit{WidgetName} and passes it to its parent widget \textit{ParentName} for management. This combines the predicates xtCreateWidget/5 and xtManageChild/1.

\textit{WidgetName} defines the name under which the widget is known in IF/Prolog. If \textit{WidgetName} is a variable, then IF/Prolog assigns a unique name and unifies \textit{WidgetName} with this name.

\textit{ToolkitName} defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as \textit{WidgetName}.

\textit{ClassName} is the name of a widget class which is not a Shell class. For these, the predicates xtCreatePopupShell/5 or xtCreateApplicationShell/4 can be used.

\textit{ParentName} specifies the parent widget of the \textit{WidgetName} widget.

\textit{ArgList} contains the resource definitions which are to apply for the \textit{WidgetName} widget.

### Arguments

- **WidgetName**: Atom, the name of a widget
- **ToolkitName**: Atom
- **ClassName**: Atom, a defined class name
- **ParentName**: Atom, the name of a widget
- **ArgList**: A list of value assignments for resources. These value assignments have the form: [\textit{ResourceName} = \textit{ResourceValue}, ...] or [\textit{ResourceName}(\textit{ResourceValue}), ...]

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(unused_widget_name)**
  - The specified widget name is already in use.

- **domain_error(internal_class_code)**
  - The specified argument must be the name of a valid widget class.

Further exceptions result if \textit{ArgList} is incorrect. These are listed under the xtSetValues/2 predicate.
See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtCreateWidget/5, xtManageChild/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtCreateManagedWidget()</td>
</tr>
</tbody>
</table>
Create pop-up shell

xtCreatePopupShell/5 creates a pop-up shell \textit{WidgetName} of the widget class \textit{ClassName}.

\textit{WidgetName} defines the name under which the widget is known in IF/Prolog. If \textit{WidgetName} is a variable, then IF/Prolog assigns a unique name and unifies \textit{WidgetName} with this name.

\textit{ToolkitName} defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as \textit{WidgetName}.

\textit{ClassName} is the name of a widget class which is not a Shell class. For these, the predicates xtCreatePopupShell/5 or xtCreateApplicationShell/4 can be used.

\textit{ParentName} specifies the parent widget of the \textit{WidgetName} widget.

\textit{ArgList} contains the resource definitions which are to apply for the \textit{WidgetName} widget.

\begin{table}[h]
\begin{tabular}{|l|l|}
\hline
Argument & Description \\
\hline
\textbf{WidgetName} & Atom, the name of a widget \\
\textbf{ToolkitName} & Atom \\
\textbf{ClassName} & Atom, a defined Shell class name \\
\textbf{ParentName} & Atom, the name of a widget \\
\textbf{ArgList} & A list of value assignments for resources. \text{These value assignments have the form:} \{\text{ResourceName = ResourceValue, ...}\} \text{ or} \{\text{ResourceName(ResourceValue), ...}\} \\
\hline
\end{tabular}
\end{table}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\begin{itemize}
\item \textbf{domain\_error(unused\_widget\_name)}: The specified widget name is already in use.
\item \textbf{domain\_error(internal\_class\_code)}: The specified argument must be the name of a valid widget class.
\end{itemize}

Further exceptions result if \textit{ArgList} is incorrect. These are listed under the xtSetValues/2 predicate.
Predicates

Predicates xtCreatePopupShell/5

See also

IF/Prolog xtSetValues/2, xtCreateApplicationShell/4
X XtCreatePopupShell()
xtCreateWidget/5 creates a widget, WidgetName.

WidgetName defines the name under which the widget is known in IF/Prolog. If WidgetName is a variable, then IF/Prolog assigns a unique name and unifies WidgetName with this name.

ToolkitName defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as WidgetName.

ClassName is the name of a widget class which is not a Shell class. For these, the predicates xtCreatePopupShell/5 or xtCreateApplicationShell/4 can be used.

ParentName specifies the parent widget of the WidgetName widget.

ArgList contains the resource definitions which are to apply for the WidgetName widget.

Arguments

- **WidgetName**: Atom, the name of a widget
- **ToolkitName**: Atom
- **ClassName**: Atom, a defined class name
- **ParentName**: Atom, the name of a widget
- **ArgList**: A list of value assignments for resources. These value assignments have the form: [ResourceName = ResourceValue, ...] or [ResourceName(ResourceValue), ...]

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(unused_widget_name)**
  The specified widget name is already in use.
- **domain_error(internal_class_code)**
  The specified argument must be the name of a valid widget class.

Further exceptions result if ArgList is incorrect. These are listed under the xtSetValues/2 predicate.
See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtCreateManagedWidget/5, xtSetValues/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtCreateWidget()</td>
</tr>
</tbody>
</table>
Destroy widget

```prolog
xtDestroyWidget ( +WidgetName )
```

xtDestroyWidget/1 deletes the widget named `WidgetItemName` and all its children.

The actual destruction is not executed until all processing steps initiated by the current event have been completed. It will then only be executed immediately if `xtDestroyWidget/1` was not called by a callback, or if there are no more outstanding callback predicates awaiting processing.

Destroying a widget causes the callback predicates specified in the destroyCallback list to be called.

**Arguments**

- **WidgetItemName** Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog `xtCreateWidget/5`
- X `XtDestroyWidget()`
Define width and height of a display

\[
\begin{align*}
\text{xtDisplayHeight} &\quad (\text{+WidgetName, ?Height}) \\
\text{xtDisplayHeightMM} &\quad (\text{+WidgetName, ?Height}) \\
\text{xtDisplayWidth} &\quad (\text{+WidgetName, ?Width}) \\
\text{xtDisplayWidthMM} &\quad (\text{+WidgetName, ?Width})
\end{align*}
\]

These predicates can be used to match the geometry of a user interface, i.e. its position and size, to the screen which is being used.

The dimensions are specified in units of either pixels or millimeters.

\text{WidgetName} is used to specify the screen.

\(\text{xtDisplayHeight}/2\) returns \textit{Height}, which is the height of the screen in pixels.

\(\text{xtDisplayHeightMM}/2\) returns \textit{Height}, which is the height of the screen in millimeters.

\(\text{xtDisplayWidth}/2\) returns \textit{Width}, the width of the screen in pixels.

\(\text{xtDisplayWidthMM}/2\) returns \textit{Width}, the width of the screen in millimeters.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Width</td>
<td>Integer</td>
</tr>
<tr>
<td>Height</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

\(X\) \quad \text{XDisplayHeight(), ...MM(), ...Width(), ...MM()}
Draw circular or elliptical arcs

xtDrawArc/7, xtDrawArcs/2

**Predicates**

- **xtDrawArc** (`+DrawWidget, +XPosition, +YPosition, +Width, +Height, +FromAngle, +ToAngle`)
- **xtDrawArcs** (`+DrawWidget, +ArcList`)

These predicates draw circular or elliptical arcs in the `DrawWidget` widget.

**xtDrawArc/7** draws a circular or elliptical arc. The arc is specified by a bounding rectangle and two angles. The centre point of the bounding rectangle is also the centre point of the circle or the ellipse of which the arc forms a part.

- **Xposition** and **YPosition** are the X and Y coordinates of the top left corner of the rectangle, relative to the coordinate system of the `DrawWidget` widget, the origin of this widget being in its top left corner.
- **Width** and **Height** specify the width and height of the rectangle in pixels. If the width and height are equal, the arc will be part of a circle.
- To permit the start and end of the arc to be defined, the full circle or ellipse is divided into units of 64ths of 360 degrees. Compared to the pointer on a clock, the origin is at three o’clock, the position 90 x 64 corresponds to 12 o’clock, 180 x 64 to 9 o’clock, etc.
- **FromAngle** defines the start position of the required arc relative to the ”3 o’clock position”. **ToAngle** defines the end position relative to the start position. A positive value indicates that the arc is to be drawn in a counterclockwise direction from the start to end position. For a negative value, the arc is drawn in a clockwise sense.

**xtDrawArcs/2** draws a number of circular or elliptical arcs according to the specifications in **ArcList**. **ArcList** contains the specifications for **XPosition, YPosition, Width, Height, FromAngle** and **ToAngle**.

**Arguments**

- **DrawWidget** Atom, the name of a Draw widget
- **XPosition** Integer
- **YPosition** Integer
- **Width** Integer ≥ 0
- **Height** Integer ≥ 0
- **FromAngle** Integer
- **ToAngle** Integer
- **ArcList** A list of lists. Each of these consists of six integers. [XPosition, YPosition, Width, Height, FromAngle, ToAngle]
Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtDrawFillArc/7</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XDrawArc(), ...s()</td>
</tr>
</tbody>
</table>
Obtain dimensions of a bitmap

\textbf{xtDrawBitmapExtents/4} \hspace{1em} \textbf{Predicates}

xtDrawBitmapExtents/4 returns the \textit{Width} and \textit{Height}, the dimensions (in pixels) of the bitmap specified by \textit{Bitmap}.

\textbf{Arguments}

- \textbf{DrawWidget} \hspace{1em} Atom, the name of a Draw widget
- \textbf{Bitmap} \hspace{1em} Atom: bitmap0 | bitmap1 ... | bitmap9
- \textbf{Width} \hspace{1em} Integer > 0
- \textbf{Height} \hspace{1em} Integer > 0

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

- IF/Prolog \hspace{1em} xtDrawFontExtents/4, xtDrawStringExtents/5
xtDrawClear/1 clears the entire screen contents of the DrawWidget widget.

Arguments

- DrawWidget: Atom, the name of a Draw widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog: xtDrawClearArea/6, xtDrawRepaint/1
- X: XClearWindow()
**xtDrawClearArea/6**

**Predicates**

Clear a window area

<table>
<thead>
<tr>
<th>xtDrawClearArea ( +DrawWidget, +XPosition, +YPosition, +Width, +Height, +Exposure )</th>
</tr>
</thead>
</table>

xtDrawClearArea/6 clears a rectangular area in the *DrawWidget* widget. This rectangle is defined by the X and Y coordinates, *XPosition* and *YPosition*, of its top left corner, and by its *Width* and *Height*.

*Exposure* specifies whether expose events will be generated to make visible areas which were hidden in the background.

**Arguments**

- **DrawWidget**: Atom, the name of a Draw widget
- **XPosition**: Integer
- **YPosition**: Integer
- **Width**: Integer > 0
- **Height**: Integer > 0
- **Exposure**: Atom: true | false | yes | no | on | off

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xtDrawClear/1
- X: XClearArea()
xtDrawCopyArea/7 copies a rectangular area of DrawWidget's window to another position in the same window. SrcXPosition and SrcYPosition specify the X and Y coordinates of the top left corner of the source rectangle. Width and Height give the width and height of this source rectangle. DestXPosition and DestYPosition give the X and Y coordinates of the destination rectangle.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type/Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawWidget</td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td>SrcXPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>SrcYPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>Width</td>
<td>Integer &gt; 0</td>
</tr>
<tr>
<td>Height</td>
<td>Integer &gt; 0</td>
</tr>
<tr>
<td>DestXPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>DestYPosition</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtDrawCopyBitmap/4
X XCopyArea()
xtDrawCopyBitmap/4 copies the bitmap specified by Bitmap from the widget DrawWidget to another position in the same widget. XPosition and YPosition specify the X and Y coordinates of the top left corner of the destination rectangle.

Arguments

- **DrawWidget** Atom, the name of a Draw widget
- **Bitmap** Atom: bitmap0 | bitmap1 ... | bitmap9
- **XPosition** Integer
- **YPosition** Integer

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog xtDrawCopyArea/7
- X XCopyArea()
Delete a line

xtDrawDeleteLine ( +DrawWidget, +XStart, +YStart, +XEnd, +YEnd )

If the saveOpList resource is switched on, the drawing operations of the DrawWidget will be saved in a history list.

xtDrawDeleteLine/5 deletes the first entry in this list which corresponds to the passed parameters XStart, YStart, XEnd and YEnd.

The deletion has no immediate effect on the contents of the screen. Only when the screen contents are redrawn will the line cease to be displayed.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawWidget</td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td>XStart</td>
<td>Integer</td>
</tr>
<tr>
<td>YStart</td>
<td>Integer</td>
</tr>
<tr>
<td>XEnd</td>
<td>Integer</td>
</tr>
<tr>
<td>YEnd</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtDrawLine/5
Draw filled circular or elliptical arcs

xtDrawFillArc/7, xtDrawFillArcs/2

These predicates draw filled circular or elliptical arcs in the DrawWidget widget. The pattern of the infill is determined by the graphics context specification.

xtDrawFillArc/7 draws a filled circular or elliptical arc. The arc is specified by a bounding rectangle and two angles. The centre point of the bounding rectangle is also the centre point of the circle or the ellipse of which the arc forms a part.

Xposition and Yposition are the X and Y coordinates of the top left corner of the rectangle, relative to the coordinate system of the DrawWidget widget.

Width and Height specify the width and height of the rectangle in pixels. If the width and height are equal, the arc will be part of a circle.

To permit the start and end of the arc to be defined, the full circle or ellipse is divided into units of 64ths of 360 degrees. Compared to the pointer on a clock, the origin is at three o’clock, the position 90 x 64 corresponds to 12 o’clock, 180 x 64 to 9 o’clock, etc.

FromAngle defines the start position of the required arc relative to the ”3 o’clock position”. ToAngle defines the end position relative to the start position. A positive value indicates that the arc is to be drawn in a counterclockwise direction from the start to end position. For a negative value, the arc is drawn in a clockwise sense.

xtDrawFillArcs/2 draws a number of circular or elliptical arcs according to the specifications in ArgList.

Arguments

DrawWidget Atom, the name of a Draw widget
XPosition Integer
YPosition Integer
Width Integer > 0
Height Integer > 0
FromAngle Integer
ToAngle Integer
ArcList A list of lists Each of these consists of six integers. [XPosition, YPosition, Width, Height, FromAngle, ToAngle]

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).
Predicates  
xtDrawFillArc/7, xtDrawFillArcs/2

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtDrawArc/7</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XFillArc(), ...s()</td>
</tr>
</tbody>
</table>
xtDrawFillPolygon/4 draws a filled polygon in the DrawWidget widget. The pattern of the infill is determined by the graphics context specification. The vertices of the polygon are listed in PointList. The polygon will be close automatically if the last point in this list is not identical to the first.

The argument Shape can be used to optimize the performance of the fill routine. The following values may be specified for Shape.

- convex: The shape is completely convex. If a straight line is drawn joining any two arbitrary vertices of the polygon, then every point on this line lies within the polygon.
- nonconvex: There are no subdivisions in the drawing, but the shape is not completely convex, i.e. not all of the points on a line joining two vertices lie within the polygon.
- complex: The drawing is subdivided, and the individual parts are to be filled separately.

CoordMode specifies the relationship between the points in the PointList.

The following are the possible values:

- coordModeOrigin: The coordinates of each point are interpreted relative to the top left corner of the DrawWidget widget.
- coordModePrevious: Apart from the first point, the coordinates of each other point are interpreted relative to the preceding one. The first point is specified relative to the top left corner of the DrawWidget widget.

**Arguments**

- DrawWidget: Atom, the name of a Draw widget
- PointList: A list of lists Each of these consists of two integers
- Shape: Atom: convex | nonconvex | complex
- CoordMode: Atom: coordModeOrigin | coordModePrevious

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- X XFillPolygon()
Draw filled rectangle

xtDrawFillRectangle ( +DrawWidget, +XPosition, +YPosition, +Width, +Height )

xtDrawFillRectangles ( +DrawWidget, +RectangleList )

These predicates draw filled rectangles in the _DrawWidget_ widget. The pattern of the infill is determined by the graphics context specification.

_xtDrawFillRectangle/_5_ draws a filled rectangle with a width _Width_ and height _Height_. _Xposition_ and _YPosition_ are the X and Y coordinates of the top left corner of the rectangle, relative to the coordinate system of the _DrawWidget_ widget.

_xtDrawFillRectangles/_2_ draws a number of filled rectangles according to the specifications in _RectangleList_.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>DrawWidget</em></td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td><em>XPosition</em></td>
<td>Integer</td>
</tr>
<tr>
<td><em>YPosition</em></td>
<td>Integer</td>
</tr>
<tr>
<td><em>Width</em></td>
<td>Integer &gt; 0</td>
</tr>
<tr>
<td><em>Height</em></td>
<td>Integer &gt; 0</td>
</tr>
<tr>
<td><em>RectangleList</em></td>
<td>A list of lists Each of these consists of four integers: [ XPosition, YPosition, Width, Height ]</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

 IF/Prolog          xtDrawRectangle/5
 X               XFillRectangle(), ...s()
Execute output operations immediately

xtDrawFlush ( +DrawWidget )

xtDrawFlush/1 forces the immediate execution of all output operations by the X server. This predicate will rarely be used, since the queue is automatically emptied if the event loop is awaiting events. It will be required when drawing operations are to be made immediately visible, even though the system is not currently within the event loop.

Arguments

DrawWidget Atom, the name of a Draw widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

X XFlush()
Query size of a font

xtDrawFontExtents/4 returns details of the font currently selected in the graphics context which is active for the DrawWidget widget.

FontDirection is the direction in which characters are to be output. This can be either fontRightToLeft or fontLeftToRight.

Ascent is the greatest height of any character in the font, in pixels.

Descent is the greatest depth of any character in the font, in pixels.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawWidget</td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td>FontDirection</td>
<td>Atom: fontLeftToRight</td>
</tr>
<tr>
<td>Ascent</td>
<td>Integer</td>
</tr>
<tr>
<td>Descent</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtDrawBitmapExtents/4, xtDrawStringExtents/5
X XTextExtents()
Query last position of the pointer

xtDrawGetXY/3 returns the coordinates PointerXPosition and PointerYPosition which were determined and saved when the last mouse event occurred.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawWidget</td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td>PointerXPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>PointerYPosition</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtDrawQueryPointer/3
Predicates xtDrawImageString/4

Output character string with foreground and background

xtDrawImageString/4 outputs a character string String in the DrawWidget widget, with its foreground and background. XPosition and YPosition are the X and Y coordinates of the baseline on which the string is output, relative to the top left corner of the Draw widget.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawWidget</td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td>XPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>YPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>String</td>
<td>Atom</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog xtDrawString/4
- X XDrawImageString()
Draw lines

\textbf{xtDrawLine} ( +\textit{DrawWidget}, +\textit{XStart}, +\textit{YStart}, +\textit{XEnd}, +\textit{YEnd} )

\textbf{xtDrawLines} ( +\textit{DrawWidget}, +\textit{PointList}, +\textit{CoordMode} )

These predicates draw lines in the \textit{DrawWidget} widget.

\texttt{xtDrawLine/5} draws a line between the point whose coordinates are given by \textit{XStart} and \textit{YStart} and the point with coordinates \textit{XEnd} and \textit{YEnd}. The coordinates are specified relative to the top left corner of the \textit{DrawWidget} widget.

\texttt{xtDrawLines/3} draws a number of lines using the coordinates contained in the \textit{PointList}. \textit{CoordMode} specifies the relationship between the points defined in the \textit{PointList} list.

The following are the possible values:

- \textit{coordModeOrigin}: The coordinates of each point are interpreted relative to the top left corner of the \textit{DrawWidget} widget.
- \textit{coordModePrevious}: Apart from the first point, the coordinates of each other point are interpreted relative to the preceding one. The first point is specified relative to the top left corner of the \textit{DrawWidget} widget.

**Arguments**

- \textit{DrawWidget}: Atom, the name of a \textit{Draw} widget
- \textit{XStart}, \textit{XEnd}, \textit{YStart}, \textit{YEnd}: Integer
- \textit{PointList}: A list of lists Each of these consists of two integers.
- \textit{CoordMode}: Atom: \textit{coordModeOrigin} \textbar{} \textit{coordModePrevious}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \hspace{1em} \texttt{xtDrawDeleteLine/5}
- X \hspace{1em} \texttt{XDrawLine()}, \ldots()
Draw points

xtDrawPoint ( +DrawWidget, +XPosition, +YPosition )

xtDrawPoints ( +DrawWidget, +PointList, +CoordMode )

These predicates draw points in the DrawWidget widget.

xtDrawPoint/3 draws a point at the position with the coordinates XPosition and YPosition. Xposition and YPosition are specified relative to the coordinate system of the DrawWidget widget.

xtDrawPoints/3 draws a number of points according to the specifications in PointList. CoordMode specifies the relationship between the points defined in PointList. The following values are possible:

- coordModeOrigin: The coordinates of each point are interpreted relative to the top left corner of the DrawWidget widget.
- coordModePrevious: Apart from the first point, the coordinates of each other point are interpreted relative to the preceding one. The first point is specified relative to the top left corner of the DrawWidget widget.

Arguments

- DrawWidget: Atom, the name of a Draw widget
- XPosition: Integer
- YPosition: Integer
- PointList: A list of lists Each of these consists of two integers.
- CoordMode: Atom: coordModeOrigin | coordModePrevious

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

X XDrawPoint(), ..s()
Query size of a memory dump

xtDrawQueryDumpFile/3

\begin{verbatim}
xtDrawQueryDumpFile ( +FileName, ?Width, ?Height )
\end{verbatim}

xtDrawQueryDumpFile/3 returns \textit{Width} and \textit{Height}, which give the dimensions of the image held in the memory dump file \textit{FileName}.

The predicate will fail if the file cannot be opened, or if it is not in the correct format.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Atom, the name of a file</td>
</tr>
<tr>
<td>Width</td>
<td>Integer $&gt; 0$</td>
</tr>
<tr>
<td>Height</td>
<td>Integer $&gt; 0$</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

IF/Prolog xtDrawReadWindowDump/7, xtDrawWriteWindowDump/2


Query position of the pointer

\[
\text{xtDrawQueryPointer} ( \, +\text{DrawWidget}, \, ?\text{XPosition}, \, ?\text{YPosition} \, )
\]

\( \text{xtDrawQueryPointer/3} \) returns the current coordinates, \( \text{XPosition} \) and \( \text{YPosition} \), of the pointer in the \( \text{DrawWidget} \) widget, relative to the top left corner of \( \text{DrawWidget} \).

**Arguments**

- **DrawWidget** : Atom, the name of a Draw widget
- **XPosition** : Integer
- **YPosition** : Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \( \text{xtDrawGetXY/3} \)
- X \( \text{XQueryPointer()} \)
Read a memory dump

xtDrawReadWindowDump/8 reads the specified rectangle from the memory dump file File-Name and copies it to the specified destination in the Draw widget.

The predicate will generate exceptions if the file cannot be opened or is not in the correct format.

SrcXPosition and SrcYPosition, Width and Height define the coordinates, the width and the height respectively of the source rectangle; DestXPosition and DestYPosition define the coordinates of the destination rectangle.

Arguments

- DrawWidget Atom, the name of a Draw widget
- SrcXPosition Integer ≥ 0
- SrcYPosition Integer ≥ 0
- Width Integer > 0
- Height Integer > 0
- DestXPosition Integer
- DestYPosition Integer
- FileName Atom, the name of a file

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

environment_error(file_not_found)

One of the specified files could not be found.

See also

IF/Prolog xtDrawQueryDumpFile/2, xtDrawWriteWindowDump/2
Set graphics context

\texttt{xtDrawRecallGC ( +DrawWidget, +GC )}

\texttt{xtDrawRecallGC/2} sets the active graphics context for the \textit{DrawWidget} widget using the graphics context held in the graphics context store identified by \textit{GC}. A total of ten stores can be set up for graphics contexts.

**Arguments**

- \texttt{DrawWidget} \hspace{1cm} Atom, the name of a Draw widget
- \texttt{GC} \hspace{1cm} Atom: gc0 \mid gc1 \ldots \mid gc9

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \hspace{1cm} \texttt{xtDrawSetParam/3, xtDrawStoreGC/2}
Draw rectangles

\[
\text{xtDrawRectangle} \ ( +\text{DrawWidget}, +\text{XPosition}, +\text{YPosition}, +\text{Width}, +\text{Height} )
\]

\[
\text{xtDrawRectangles} \ ( +\text{DrawWidget}, +\text{RectangleList} )
\]

These predicates draw rectangles in the Draw widget.

xtDrawRectangle/5 draws a rectangle with a width of Width and height Height. The top left corner of the rectangle has the X coordinate XPosition and the Y coordinate YPosition. The coordinates are always relative to the origin of the DrawWidget widget (top left corner).

xtDrawRectangles/2 draws a number of rectangles according to the specifications defined in RectangleList.

Arguments

- **DrawWidget** Atom, the name of a Draw widget
- **XPosition** Integer
- **YPosition** Integer
- **Width** Integer $> 0$
- **Height** Integer $> 0$
- **RectangleList** A list of lists Each of these consists of four integers: [XPosition, YPosition, Width, Height]

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog xtDrawFillRectangle/5
- X XDrawRectangle(), ...s()
Predicates xtDrawRepaint/1

**Restore screen contents**

| xtDrawRepaint ( +DrawWidget ) |

xtDrawRepaint/1 clears the contents of the Draw widget’s window and generates an expose event. The effect is that the contents of the window are completely redrawn.

**Arguments**

- DrawWidget Atom, the name of a Draw widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog xtDrawClear/1
Draw line segments

xtDrawSegments/2 draws line segments in the Draw widget, according to the specifications in SegmentList; each entry in SegmentList specifies the X and Y coordinates for the start and end points of a line segment.

Using xtDrawSegments/2 it is possible to draw lines which are not necessarily continuous from start to end. Each operation links a pair of end points in the specified list. The coordinates are always relative to the origin of the Draw widget (top left corner).

Arguments

- DrawWidget: Atom, the name of a Draw widget
- SegmentList: A list of lists Each of these consists of four integers: [XStart, YStart, XEnd, YEnd]

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog: xtdrawline/5
- X: XDrawSegments()
Predicates

xtDrawSetDashes/3

Set line pattern

xtDrawSetDashes ( +DrawWidget, +Offset, +DashList )

xtDrawSetDashes/3 sets the line style for displaying broken lines in the Draw widget’s graphics context. DashList contains the pattern for the line. This line style is defined by a list of integers, specifying which of the pixels in a row of pixels will be drawn in, and which will not. The first, third, fifth number, etc., define the pixels which are to be drawn; the second, fourth, sixth numbers, etc., define the gaps. The actual line will be drawn by repeating this line pattern as often as necessary to make up the length of the line.

Offset defines the start point, and hence the way that the pattern is repeated, by specifying the point in the list where the line style that is currently to be output should begin.

Arguments

- DrawWidget: Atom, the name of a Draw widget
- DashList: A list of integers
- Offset: Integer \( \geq 0 \)

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog: xtDrawLine/5
- X: XSetDashes()
Set parameters for the graphics context

xtDrawSetParam/3 sets the parameters for the current graphics context for the Draw widget. 

*Which* specifies the component for which the specification is to be made, *GCValue* contains the actual specification.

The valid parameter specifications and values are:

<table>
<thead>
<tr>
<th>Which</th>
<th>Permissible values for GCValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>drawFillStyle</td>
<td>fillSolid, fillTiled, fillStippled, fillOpaqueStippled</td>
</tr>
<tr>
<td>drawFillRule</td>
<td>evenOddRule, windingRule</td>
</tr>
<tr>
<td>drawSetArcMode</td>
<td>arcPieSlice, arcChord</td>
</tr>
<tr>
<td>drawResourceForeground</td>
<td>color0 .. color9</td>
</tr>
<tr>
<td>drawResourceBackground</td>
<td>color0 .. color9</td>
</tr>
<tr>
<td>drawTileForeground</td>
<td>color0 .. color9</td>
</tr>
<tr>
<td>drawTileBackground</td>
<td>color0 .. color9</td>
</tr>
<tr>
<td>drawLineWidth</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>drawLineStyle</td>
<td>lineSolid, lineOnOffDash, lineDoubleDash</td>
</tr>
<tr>
<td>drawJoinStyle</td>
<td>joinMiter, joinRound, joinBevel</td>
</tr>
<tr>
<td>drawCapStyle</td>
<td>capNotLast, capButt, capRound, capProjecting</td>
</tr>
<tr>
<td>drawResourceStipple</td>
<td>bitmap0 .. bitmap9</td>
</tr>
<tr>
<td>drawResourceTile</td>
<td>bitmap0 .. bitmap9</td>
</tr>
<tr>
<td>drawSubwindowMode</td>
<td>clipByChildren, includeInferiors</td>
</tr>
</tbody>
</table>

- **drawFillStyle**: specifies the pattern to be used in filling graphic elements.
- **drawFillRule**: specifies the areas in a filled polygon which are to be covered with the infill pattern, and which not.
- **drawSetArcMode**: specifies the way in which a circular or elliptical arc is to be filled with the infill pattern.
- **drawResourceFont**: specifies which font is currently in use. The 'font' component is determined by the resources of the DrawWidget widget. *GCValue* specifies which of the fonts, font0 to font9, to use.
- **drawResourceForeground**: specifies which color is used as the foreground color. The 'foreground' component is determined by the resources of the DrawWidget widget. *GCValue* specifies which of the colors, color0 to color9, to use.
- **drawResourceBackground**: specifies which color is used as the background color. The 'background' component is determined by the resources of the DrawWidget widget. *GCValue* specifies which of the colors, color0 to color9, to use.
Predicates xtDrawSetParam/3

- **drawTileForeground**: specifies which foreground is to be used when the grid for the graphics context is being built up from a bitmap. The new foreground for the grid will not be used until xtDrawSetParam/3 is called again, with the value xtDrawSetResourceTile for Which GCValue specifies which of the colors, color0 to color9, to use.

- **drawTileBackground**: specifies which background is to be used when the grid for the graphics context is being built up from a bitmap. The new background for the grid will not be used until xtDrawSetParam/3 is called again, with the value xtDrawSetResourceTile for Which GCValue specifies which of the colors, color0 to color9, to use.

- **drawLineWidth**: specifies the thickness of the lines.

- **drawLineStyle**: specifies the style of the lines.

- **drawJoinStyle**: specifies how lines are joined together when they have the same start or end points.

- **drawCapStyle**: specifies how the end of a line is drawn.

- **drawResourceStipple**: specifies which bitmap to use as the pattern for level 1 of the pixmap. The 'stipple' component is determined by the resources of the DrawWidget widget. GCValue specifies which bitmap to use, bitmap0 to bitmap9.

- **drawResourceTile**: specifies which bitmap to use as the pattern. The 'tile' component is determined by the resources of the DrawWidget widget. The pixmap for the pattern is built up from the bitmap. In doing this, the foreground and background colors are used, as previously set using xtDrawSetParam/3, when Which was set either to xtDrawTileForeground or to xtDrawTileBackground. If no colors have been explicitly specified, the initial foreground or background color of the Draw widget will be used, as applicable. GCValue specifies which bitmap to use, bitmap0 to bitmap9.

- **drawSubwindowMode**: defines the subwindow_mode component of the graphics context. clipByChildren specifies that graphics operations are truncated by all visible children; includeInferior specifies that they are drawn through all subwindows.

**Arguments**

**DrawWidget** 
Atom, the name of a Draw widget

**Which** 
Atom: drawFillStyle | drawFillRule | drawSetArcMode | drawResourceColor | drawResourceFont | drawResourceForeground | drawResourceBackground | drawTileForeground | drawTileBackground | drawLineWidth | drawLineStyle | drawJoinStyle | drawCapStyle | drawResourceStipple | drawResourceTile | drawSubwindowMode

**GCValue** 
Term, depends on the Which argument (see table)
Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(parameter_type)**
  An invalid value has been specified for the Which parameter.

- **domain_error(fillStyle)**
  The specified argument must be one of the atoms fillSolid, fillTiled, fillStippled or fillOpaqueStippled.

- **domain_error(fillRule)**
  The specified argument must be one of the atoms evenOddRule or windingRule.

- **domain_error(arcMode)**
  The specified argument must be one of the atoms arcChord or arcPieSlice.

- **domain_error(lineStyle)**
  The specified argument must be one of the atoms lineSolid, lineOnOffDash or lineDoubleDash.

- **domain_error(joinStyle)**
  The specified argument must be one of the atoms joinMiter, joinRound or joinBevel.

- **domain_error(capStyle)**
  The specified argument must be one of the atoms capNotLast, capButt, capRound or capProjecting.

- **domain_error(fontX)**
  The specified argument must be one of the atoms font0, ... font9.

- **domain_error(colorX)**
  The specified argument must be one of the atoms color0, ... color9.

- **domain_error(bitmapX)**
  The specified argument must be one of the atoms bitmap0, ... bitmap9.

See also

IF/Prolog
- xtDrawRecallGC/2, ...SetDashes/3, ...SetRubberGC/1, ...StoreGC/2

X
- XSetBackground(), XSetFillRule(), XSetFillStyle(), XSetFont(), XSetForeground(), XSetStipple(), XSetTile()
Set rubber-banding mode

xtDrawSetRubberGC (+DrawWidget)

xtDrawSetRubberGC/1 sets the rubber-banding mode in the graphics context of the Draw widget.

Rubber-banding is a special mode for drawing elastic outlines of objects. Any object drawn in this mode appears inverted on the screen; as a result, a repeat of the identical drawing procedure will cancel out the first one. This mode enables temporary drawings of objects to be produced, generally the outlines of graphic elements. This makes it possible to represent the movement of graphic objects across the screen.

The pixel values for the drawing and the foreground are adjusted such that, provided they have the same values, the old pixel values of the drawing will be used. If they have different values, the result will be that the Draw widget’s foreground color is used for drawing in those areas which have the Draw widget’s background color, and vice versa. The color which will result if this operation covers a color which is neither the foreground color nor the background one is undefined. It is guaranteed that the color of the original drawing will be restored.

Arguments

DrawWidget Atom, the name of a Draw widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtDrawUnsetRubberGC/1
Set origin for the pattern

**xtDrawSetTSOrigin** ( +DrawWidget, +XPosition, +YPosition )

xtDrawSetTSOrigin/3 sets the coordinates, \textit{XPosition} and \textit{YPosition}, of the initial point, starting from which a pattern will be drawn. The start point is calculated relative to the top left corner of the Draw widget.

**Arguments**

- **DrawWidget** Atom, the name of a Draw widget
- **XPosition** Integer
- **YPosition** Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- X XSetTSOrigin()
Save graphics context

\texttt{xtDrawStoreGC ( +DrawWidget, +GC )}

xtDrawStoreGC/2 saves the active graphics context for the \textit{DrawWidget} widget in the graphics context store identified by \textit{GC}. A total of ten stores can be set up for graphics contexts.

\textbf{Arguments}

\begin{itemize}
  \item \texttt{DrawWidget} Atom, the name of a Draw widget
  \item \texttt{GC} Atom: gc0 | gc1 ... | gc9
\end{itemize}

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

\begin{itemize}
  \item IF/Prolog \texttt{xtDrawRecallCG/2}
\end{itemize}
Output a character string

```
xtdrawstring ( +drawwidget, +xposition, +yposition, +string )
```

`xtdrawstring/4` outputs the character string `string` in the Draw widget. `xposition` and `yposition` are the X and Y coordinates of the baseline on which the string is output, relative to the top left corner of the Draw widget.

**Arguments**

- `drawwidget` Atom, the name of a Draw widget
- `xposition` Integer
- `yposition` Integer
- `string` Atom

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog `xtdrawimagestring/4`
- X `Xdrawstring()`
Query the size of a character string

xtDrawStringExtents/5 returns details about the size of the character string String in the graphics context which is active for the Draw widget.

*Width* is the width of the character string in pixels.

*Ascent* is the greatest height of any character in the font, in pixels.

*Descent* is the greatest depth of any character in the font, in pixels.

**Arguments**

- **DrawWidget** Atom, the name of a Draw widget
- **String** Atom
- **Width** Integer
- **Ascent** Integer
- **Descent** Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog xtDrawBitmapExtents/4, xtDrawFontExtents/4
- X XTextExtents()
xtDrawUnsetRubberGC/1

**Cancel rubber-banding mode**

xtDrawUnsetRubberGC ( +DrawWidget )

xtDrawUnsetRubberGC/1 cancels the rubber-banding mode in the graphics context of the Draw widget.

**Arguments**

- **DrawWidget** Atom, the name of a Draw widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog xtDrawSetRubberGC/1
Predicates xtDrawWriteWindowDump/2

Create memory dump of the window contents

| xtDrawWriteWindowDump ( +DrawWidget, +FileName ) |

xtDrawWriteWindowDump/2 writes a memory dump of the Draw widget’s window into the file named FileName.

Arguments

- DrawWidget: Atom, the name of a Draw widget
- FileName: Atom, the name of a file

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- environment_error(could_not_open_file)
  The specified file could not be opened for writing.
- environment_error(dump_failed)
  The predicate could not be successfully executed.

See also

IF/Prolog xtDrawReadWindowDump/7, xtDrawQueryDumpFile/2
Output a compound string

xtDrawXmString/8 outputs a compound string XmString in a Draw widget.

The arguments XPosition, YPosition and Width define the rectangle within which the compound string should be output. These coordinates should be specified relative to the top left corner of the Draw widget.

Alignment specifies the justification of the compound string.

LayoutDirection specifies the direction in which the individual segments of the compound string are output, and thus also determines the exact meaning of Alignment. This argument does not affect the sequence of the characters within a segment of text.

If ClipRectangleList is not the empty list, then it specifies rectangles which further restrict the area which is drawn.

Arguments

DrawWidget Atom, the name of a Draw widget
XmString Atom, or a list, a valid XmString
XPosition Integer
YPosition Integer
Width Integer > 0
Alignment Atom: alignment_center | alignment_beginning | alignment_end
LayoutDirection Atom: string_direction_l_to_r | string_direction_r_to_l
ClipRectangleList A list of lists These contain the following four elements: [XPos, YPos, Width, Height]

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtDrawXmStringImage/8
OSF/Motif XmStringDraw
Output compound string with foreground and background

xtDrawXmStringImage/8 outputs a compound string \textit{XmString} in a Draw widget, with a foreground and background.

The arguments \textit{XPosition}, \textit{YPosition} and \textit{Width} define the rectangle within which the compound string should be output. These coordinates should be specified relative to the top left corner of the Draw widget.

\textit{Alignment} specifies the justification of the compound string.

\textit{LayoutDirection} specifies the direction in which the individual segments of the compound string are output, and thus also determines the exact meaning of \textit{Alignment}. This argument does not affect the sequence of the characters within a segment of text.

If \textit{ClipRectangleList} is not the empty list, then it specifies rectangles which further restrict the area which is drawn.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawWidget</td>
<td>Atom, the name of a Draw widget</td>
</tr>
<tr>
<td>XmString</td>
<td>Atom, or a list, a valid XmString</td>
</tr>
<tr>
<td>XPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>YPosition</td>
<td>Integer</td>
</tr>
<tr>
<td>Width</td>
<td>Integer &gt; 0</td>
</tr>
<tr>
<td>Alignment</td>
<td>Atom: alignment_center</td>
</tr>
<tr>
<td>LayoutDirection</td>
<td>Atom: string_direction_l_to_r</td>
</tr>
<tr>
<td>ClipRectangleList</td>
<td>A list of lists These contain the following four elements: [XPos, YPos, Width, Height]</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \texttt{xtDrawXmString/8}
- OSF/Motif \texttt{XmStringDrawImage()}

Motif Manual 446 IF/Prolog V5.2
Output underlined compound string

xtDrawXmStringUnderline ( +DrawWidget, +XmString, +XPosition, +YPosition, +Width, +Alignment, +LayoutDirection, +ClipRectangleList, +SubXmStringToBeUnderlined )

xtDrawXmStringUnderline/9 outputs a partly underlined compound string XmString in a Draw widget.

The arguments XPosition, YPosition and Width define the rectangle within which the compound string should be output. These coordinates should be specified relative to the top left corner of the Draw widget.

Alignment specifies the justification of the compound string.

LayoutDirection specifies the direction in which the individual segments of the compound string are output, and thus also determines the exact meaning of Alignment. This argument does not affect the sequence of the characters within a segment of text.

If ClipRectangleList is not the empty list, then it specifies rectangles which further restrict the area which is drawn.

SubXmStringToBeUnderlined defines which partial string within XmString is to be underlined.

Arguments

- DrawWidget Atom, the name of a Draw widget
- XmString Atom, or a list, a valid XmString
- XPosition Integer
- YPosition Integer
- Width Integer > 0
- Alignment Atom: alignment_center | alignment_beginning | alignment_end
- LayoutDirection Atom: string_direction_l_to_r | string_direction_r_to_l
- ClipRectangleList A list of lists These contain the following four elements: [XPos, YPos, Width, Height]
- SubXmStringToBeUnderlined Atom, or a list, a valid XmString

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- OSF/Motif XmStringDrawUnderline()
Flush the output buffer of the X server

**xtFlush/1**

xtFlush/1 flushes the output buffer of the X server assigned to the *WidgetName* widget.

**Arguments**

*WidgetName* Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Hints**

It is normally not necessary to explicitly flush the buffer. However, if the screen is to be updated during a lengthy calculation, this predicate is useful.

**See also**

*X* XFlush()
Get action cause

\texttt{\texttt{xtGetActionKeysym} ( ?Keysym, ?Modifiers )}

\texttt{xtGetActionKeysym/2} can be used to determine which keys were pressed to activate an action routine if used from within the action predicate.

\texttt{xtGetActionKeysym/2} retrieves the \textit{Keysym} and \textit{Modifiers} that matched the event specification in the translation table entry.

\textbf{Arguments}

\begin{itemize}
  \item \textbf{Keysym} \hspace{1cm} Atom, Keysym or ’noSymbol’
  \item \textbf{Modifiers} \hspace{1cm} A list of atoms, or an empty list
\end{itemize}

List elements: anyModifier \mid shift \mid control \mid lock \mid mod1 \mid mod2 \mid mod3 \mid mod4 \mid mod5

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

\begin{itemize}
  \item IF/Prolog \hspace{1cm} \texttt{xtKeysymToKeycodeListe/4}
  \item X \hspace{1cm} XtGetActionKeysym()
\end{itemize}
Predicates xtGetApplicationResources/3

Retrieve values of resources

xtGetApplicationResources/3 returns the values for general resources of the application. These resources can be used to configure an application, for example by means of resource files. The general resources concerned are not associated with particular widgets. The specific widget resources are automatically provided in the corresponding Create functions.

The Resource Manager proceeds as follows to determine the resource values:

1. If the resource is defined in ArgList, then this value will be returned.

2. If the resource is contained in the Resource Manager’s database, then this value will be converted to a IF/Prolog term, and returned as the result. The database concerned is created from the entries in the resource files, or from the resource value definitions in the command line when the application is called up.

3. If neither of the above cases applies, then the default value specified in the ResourceList will be used.

Values which are obtained from the Resource Manager’s database must be converted from a string representation (e.g. in a resource file) into a IF/Prolog term. If the first character is a digit, then an integer will be generated, in all other cases an atom.

The widget named WidgetName may be any widget, but is generally the toplevel widget of the application. It is used to identify the application to the Resource Manager.

ResourceList specifies the resources which are to be provided, and the associated default values. For each resource which is to be interrogated, the ResourceList must contain a list with four elements in it:

[Name, Class, Result, Default]

Name Atom, the name of the resource
Class Atom, the class of the resource
Result Term, with which the result will be unified (generally a variable)
Default Term, default value for the resource

ArgList specifies a list of value assignments for the resources, and will be referred to in determining the required resource values.
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>ResourceList</td>
<td>A list of lists These contain the following elements: [Name, Class, Result, Default]</td>
</tr>
<tr>
<td>ArgList</td>
<td>A list of value assignments for resources.</td>
</tr>
<tr>
<td></td>
<td>These value assignments have the form: [ResourceName = ResourceValue, ...] or [ResourceName(ResourceValue), ...]</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\[domain\_error(resource\_value\_pair)\]

The specified list elements must be value assignments.

See also

X

XtGetApplicationResources()
Retrieves name and class of an application

\texttt{xtGetApplicationNameAndClass} \( ( + \text{WidgetName}, ?\text{ApplicationName}, \)
\( \ ?\text{ApplicationClass} ) \)

\texttt{xtGetApplicationNameAndClass/3} retrieves the name and class of an application. The name \textit{ApplicationName} and the class \textit{ApplicationClass} are the strings the resource manager uses to look up resources for the application and its widgets.

\textbf{Arguments}

- \texttt{WidgetName} \hspace{1cm} Atom, the name of a widget
- \texttt{ApplicationName} \hspace{1cm} Atom, the name of the application
- \texttt{ApplicationClass} \hspace{1cm} Atom, the class name of the application

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

- IF/Prolog \hspace{1cm} \texttt{xtGetDefault/3}
- X \hspace{1cm} \texttt{XtGetApplicationNameAndClass()}
xtGetDefault/3 accesses the resource database for an entry that matches Name and Option; i.e. a line that contains both Name and Option will be found. Name will usually be the name of the application.

**Arguments**

- **Name**: Atom, name or class of an application
- **Option**: Atom, name of an option
- **ResourceValue**: Atom, value of a resource

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Example**

Assume the file `.Xdefaults` was loaded into the database and contains the line:

```
Mwm.borderWidth: 3
```

```
[user] ?- xtGetDefault('Mwm',borderWidth,Value),nl.<ENTER>
Value = '3'
```

yes

**See also**

- X XGetDefault()
Retrieve color information

\[
\text{xtGetDefaultColorInfo( } \text{?ColorType, } \text{?DefaultDepth })
\]

xtGetDefaultColorInfo/2 retrieves color information from the default screen of the display. First the default depth is retrieved and then a supported color type for this depth is looked for. Even if more than one color type is supported xtGetDefaultColorInfo/3 returns only the first match. Color types are looked for in the order \text{directColor}, \text{trueColor}, \text{pseudoColor}, \text{staticColor}, \text{grayScale} or \text{staticGray}. If no color type can be found, \text{none} is returned.

Arguments

\begin{itemize}
  \item ColorType : Atom: \text{directColor} \mid \text{trueColor} \mid \text{pseudoColor} \mid \text{staticColor} \mid \text{grayScale} \mid \text{staticGray} \mid \text{none}
  \item DefaultDepth : Integer
\end{itemize}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

\begin{itemize}
  \item X \quad \text{XMatchVisualInfo()}
\end{itemize}
Retrieves event component

**xtGetEventComponent ( ?EventComponent )**

**xtGetEventComponents ( ?EventComponentList )**

These predicates supply the components of the event structure for the last event.

*xtGetEventComponent/1* returns the component *EventComponent* of the last event, or the one currently reported.

*xtGetEventComponents/1* can be used to interrogate a number of components of the event structure of the last event.

The components of the event structure are converted to IF/Prolog representation only upon request, and must therefore be explicitly interrogated.

In most cases it is not absolutely necessary for an application to interrogate such components, because the fact that a special callback predicate has been called with the appropriate details in itself provides a great deal of information about the event.

*EventComponent* is an expression of the form

```
Component = Value or Component(Value)
```

In this, *Component* specifies which component is to be interrogated and *Value* will be unified with the component value.

*EventComponentList* is a list of terms of the same form as *EventComponent*.

The application should first check the *eventType*. The permissible components of the event structure depend on this. The IF/Prolog names for event components are the names of the C structure components, prefixed by ‘event’. The start of each new word is indicated by the use of an uppercase letter, not by an underline character. However, where the structure names are used for more than one purpose, for example the name ‘detail’, these are modified when used in component specifications so that there is no ambiguity.

```
\[ x \rightarrow \text{eventX} \\
\text{detail} \rightarrow \text{eventStackDetail or eventInOutDetail} \]
```

If *eventType* has the value *noEvent*, then either no event has so far been reported, or the last callback was reported by a special event which is outside the X system, e.g. a timer event.

The appendix contains a list of all the valid event structure components.

**Arguments**

- **EventComponent**: Value assignment for a component of an event structure
- **EventComponentList**: A list of value assignments for components of an event structure
Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**domain_error(component_list)**

The elements specified in the list must be value assignments.

**domain_error(type_component)**

The specified structure component is not permitted for this event structure.

**domain_error(event_component)**

The structure component which has been specified is not known to the Motif interface.

**system_error(no_event_to_analyze)**

The xtGetEventComponent(s)/1 predicate may only be called from within a callback.
Get multiclick Time

xtGetMultiClickTime/2 retrieves the time lap in milliseconds that will serve to determine whether a multi-click or just a repeated click has occurred.

The translation manger uses this time interval to determine if multiple events are to be interpreted as a repeated event for purposes of matching a translation entry containing a repeat count.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>MultiClickTime</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog xtSetMultiClickTime/2
- X XtGetMultiClickTime()
Retrieve values of resources

\[ xtGetValue ( +WidgetName, +ArgValue ) \]

\[ xtGetValues ( +WidgetName, +ArgList ) \]

These predicates supply the values of widget resources. These values will be converted to the appropriate IF/Prolog format.

\texttt{xtGetValue/2} returns the value of a resource of the widget named \textit{WidgetName}.

\texttt{xtGetValues/2} returns the value of a number of resources of the widget named \textit{WidgetName}.

The resources to be queried are specified in the following form:

\begin{itemize}
  \item \textbf{ScrollBar} = 50%
  \item \textbf{PageSize} (2000)
\end{itemize}

\textbf{ScrollBar} specifies which resource is being queried.

\textbf{PageSize} will be unified with the current resource value.

**Arguments**

- \textbf{WidgetName} Atom, the name of a widget
- \textbf{ArgValue} Value assignment for a resource. This value assignment has the form: ResourceName = ResourceValue or ResourceName(ResourceValue).
- \textbf{ArgList} A list of value assignments for resources. These value assignments have the form: \{ResourceName = ResourceValue, \ldots\} or \{ResourceName(ResourceValue), \ldots\}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \texttt{domain_error(resource_name)} The specified resource identifier does not exist.
- \texttt{domain_error(resource)} The specified resource is not valid for this widget.
- \texttt{existence_error(resource_type)} One of the specified resources cannot be queried by IF/Prolog.
Hints

There are certain resources which cannot be converted to IF/Prolog format. In these cases, the system returns opaque atoms which start with xtArgVal_, instead of the values. These can then be specified in a subsequent call of xtSetValue/2. This makes it possible to pass resource values from one widget to another, even though they cannot be converted to a meaningful IF/Prolog representation.

See also

| IF/Prolog | xtSetValue/2, xtSetValues/2 |
| X | XtGetValue(), XtGetValues() |
Predicates

xtGetWidgetMapState/2

**Determine whether widget is mapped**

```
xtGetWidgetMapState ( +WidgetName, ?MapState )
```

xtGetWidgetMapState/2 returns the map state of the window of `WidgetName`.

The predicate accesses the internal attributes structure of the window of `WidgetName` and returns its map state. A widget will be actually visible on the screen if its window and all ancestors have been mapped, the window being not obscured by sibling windows, and the window manager having processed the mapping request.

The map state 'isUnviewable' is returned if the window of `WidgetName` is mapped but some ancestor is not.

**Arguments**

- **WidgetName** Atom, the name of a widget
- **MapState** Atom: isUnmapped | isViewable | isUnviewable

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- X XGetWindowAttributes()
Reserve buttons and redirect button events

xtGrabButton/9 redirects button events to *WidgetName*’s window. This may be useful if the application wants e.g. a 'buttonRelease' event to be reported to the window where the 'buttonPress' event occurred, independently of where the pointer moves meanwhile. *Button* is the button for which grabbing is to take place. *Modifiers* restrict grabbing to the occasions when the modifier keys given in the list are pressed.

All button events for *Button* that occur outside the application are reported to *WidgetName*’s window. If *OwnerEvents* is true this is also true for button events inside the application. Otherwise button events inside the application’s windows are reported normally.

If *PointerMode* or *KeyboardMode* is set to grabModeSync no event from the respective device is processed until the grab is released. With ‘grabModeAsync’ processing continues as usual.

The pointer is not allowed to leave the window of the *ConfineTo* widget. If you wish the pointer to be free to move anywhere on the screen you may use none.

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **Button**: Atom: anyButton | button1 | ... | button5
- **Modifiers**: A list of atoms, or an empty list
  - List elements: anyModifier | shift | control | lock | mod1 | mod2 | mod3 | mod4 | mod5
- **OwnerEvents**: Atom: true | false | yes | no | on | off
- **EventMask**: A list of atoms, or an empty list
  - List elements: noEventMask | keyPressMask | keyReleaseMask | buttonPressMask | buttonReleaseMask | enterWindowMask | leaveWindowMask | pointerMotionMask | pointerMotionHintMask | button1MotionMask | ... | button5MotionMask | buttonMotionMask | keymapStateMask | exposureMask | visibilityChangeMask | structureNotifyMask | substructureRedirectMask | focusChangeMask | propertyChangeMask | colormapChangeMask | ownerGrapButtonMask
- **PointerMode**: Atom: grabModeSync | grabModeAsync
- **KeyboardMode**: Atom: grabModeSync | grabModeAsync
- **ConfineTo**: Atom, the name of a widget or 'none'
- **Cursor**: Atom, a cursor name
  - Examples: arrow, mouse, pencil
Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**domain_error**(modifier)
The list elements which may be used for the Modifiers argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

See also

IF/Prolog xtGrabButtonboard/4, xtUngrabButton/3
X XGrabButton(), XtGrabButton()
xtGrabKey/3 reserves the keyboard of the display assigned to the WidgetName widget if the key Keycode is pressed at the same time as the key defined in Modifiers, and redirects all key events to the highest widget in the hierarchy. This reservation is removed when the key is released again.

Modifiers is one or more additional keys.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Keycode</td>
<td>Integer ≥ 0. 0 corresponds to anyKey</td>
</tr>
<tr>
<td>Modifiers</td>
<td>A list of atoms, or an empty list</td>
</tr>
</tbody>
</table>

List elements: anyModifier | shift | control | lock | mod1 | mod2 | mod3 | mod4 | mod5

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error(modifier)

The list elements which may be used for the Modifiers argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

See also

IF/Prolog  xtGrabKeyboard/4, xtUngrabKey/3
X           XGrabKey()
Reserve keyboard and redirect keyboard event

\[
\text{xtGrabKey} \ (\ +\text{WidgetName}, +\text{Keycode}, +\text{Modifiers}, +\text{OwnerEvents}, +\text{EventMask}, \\
+\text{PointerMode} +\text{KeyboardMode} )
\]

\(\text{xtGrabKey/6}\) reserves the keyboard of the display assigned to the \text{WidgetName} widget if the key \text{Keycode} is pressed at the same time as the key defined in \text{Modifiers}, and redirects all key events to the highest widget in the hierarchy. This reservation is removed when the key is released again.

\text{Modifiers} is one or more additional keys.

If \text{OwnerEvents} is true grabbing also takes place inside the own application, only outside if false.

If \text{PointerMode} or \text{KeyboardMode} is set to \text{grabModeSync} no event from the respective device is processed until the grab is released. With \text{grabModeAsync} processing continues as usual.

Arguments

- \text{WidgetName} Atom, the name of a widget
- \text{Keycode} Integer \geq 0. 0 corresponds to anyKey
- \text{Modifiers} A list of atoms, or an empty list
  - List elements: anyModifier | shift | control | lock | mod1 | mod2 | mod3 | mod4 | mod5
- \text{OwnerEvents} Atom: true | false | yes | no | on | off
- \text{PointerMode} Atom: \text{grabModeSync} | \text{grabModeAsync}
- \text{KeyboardMode} Atom: \text{grabModeSync} | \text{grabModeAsync}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\text{domain_error(modifier)}

The list elements which may be used for the \text{Modifiers} argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

See also

- IF/Prolog xtGrabKey/3, xtGrabKeyboard/4, xtUngrabKey/3
- X XGrabKey()
xtGrabKeyboard/4

Reserve keyboard

**xtGrabKeyboard** ( +WidgetName, +OwnerEvents, +PointerMode, +KeyboardMode )

xtGrabKeyboard/4 reserves the keyboard of the display assigned to the *WidgetName* widget for the application. All keyboard events are diverted to the calling application.

If *OwnerEvents* is **false**, all the keyboard events are passed on as specified by *PointerMode* and *KeyboardMode*. If *OwnerEvents* is **true**, all the keyboard events which concern this application are passed on as usual.

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **OwnerEvents**: Atom: true | false | yes | no | on | off
- **PointerMode**: Atom: grabModeSync | grabModeAsync
- **KeyboardMode**: Atom: grabModeSync | grabModeAsync

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xtGrabKey/3, xtUngrabKeyboard/1
- X: XGrabKeyboard()
Reserve pointer and redirect pointer events

xtGrabPointer/3 redirects pointer events to WidgetName’s window. Modifiers restrict grabbing to the occasions when the modifier keys given in the list are pressed.

All pointer events that occur outside the application are reported to WidgetName’s window. If OwnerEvents is true this is also true for pointer events inside the application. Otherwise pointer events inside the application’s windows are reported normally.

If PointerMode or KeyboardMode is set to grabModeSync no event from the respective device is processed until the grab is released. With grabModeAsync processing continues as usual.

The pointer is not allowed to leave the window of the ConfineTo widget. If you wish the pointer to be free to move anywhere on the screen you may use none.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Pointer</td>
<td>Atom: anyPointer</td>
</tr>
<tr>
<td>Modifiers</td>
<td>A list of atoms, or an empty list</td>
</tr>
<tr>
<td>OwnerEvents</td>
<td>Atom: true</td>
</tr>
<tr>
<td>EventMask</td>
<td>A list of atoms, or an empty list</td>
</tr>
<tr>
<td>PointerMode</td>
<td>Atom: grabModeSync</td>
</tr>
<tr>
<td>KeyboardMode</td>
<td>Atom: grabModeSync</td>
</tr>
<tr>
<td>ConfineTo</td>
<td>Atom, the name of a widget or 'none'</td>
</tr>
<tr>
<td>Cursor</td>
<td>Atom, a cursor name</td>
</tr>
</tbody>
</table>

Examples: arrow, mouse, pencil

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).
See also

IF/Prolog  xtGrabButton/9, xtGrabKeyboard/4, xtGrabKey/3, xtUngrabPointer/1

X  XGrabPointer(), XtGrabPointer()
Predicates xtHasCallbacks/2

Check existence of callbacks

\begin{verbatim}
xtHasCallbacks ( +WidgetName, +CallbackName )
\end{verbatim}

xtHasCallbacks/2 checks whether there are callback predicates registered in the callback list specified by \emph{CallbackName} for a widget named \emph{WidgetName}. This goal will succeed if callbacks are registered, otherwise it will fail.

Arguments

\begin{itemize}
  \item \texttt{WidgetName} \hspace{1cm} Atom, the name of a widget
  \item \texttt{CallbackName} \hspace{1cm} Atom, the name of a callback
\end{itemize}

See also

\begin{itemize}
  \item X \hspace{1cm} XtHasCallbacks()
\end{itemize}
Initialize Toolkit Intrinsics

\[
\text{xtInitialize} ( \ ?\text{WidgetName}, +\text{ToolkitName}, +\text{ApplicationClass}, +\text{OptionList}, \\
\quad +\text{ArgvList} )
\]

\[
\text{xtInitialize} ( \ ?\text{WidgetName}, +\text{ToolkitName}, +\text{ApplicationClass}, +\text{OptionList}, \\
\quad +\text{ArgvList} )
\]

\[
\text{xtInitialize} ( \ ?\text{WidgetName}, +\text{ToolkitName}, +\text{ApplicationClass}, +\text{OptionList}, \\
\quad +\text{ArgvList}, +\text{FallbackResources} )
\]

xtInitialize/5 initializes the Xt Toolkit Intrinsics for the application and sets up a link to the server. A widget named \text{WidgetName} belonging to the class \text{topLevelShellWidgetClass} is then set up. This widget is a child of the root window, and can be used as a parent widget for further widgets.

\text{WidgetName} defines the name under which the widget is known in IF/Prolog. If \text{WidgetName} is a variable, then IF/Prolog assigns a unique name and unifies \text{WidgetName} with this name.

\text{ToolkitName} defines the name under which the widget is known to the Toolkit’s Resource Manager. This name can be used in external definitions of resource values (in resource files). In general, this name will be the same as \text{WidgetName}.

\text{ApplicationClass} defines the class name for the application. This name can be used in resource files, to identify the application.

\text{ArgvList} is a list of arguments from the command line which are to be evaluated by Xt Toolkit. It should be noted that the first element in this list is always interpreted as the name of the application and can be used, for example, in resource files. The arguments from the Xt Toolkit command line which it is possible to evaluate will be found in the manual: \textit{Xt Toolkit (Intrinsics) Programmer’s Reference [16]}.

\text{OptionList} can be used to affect the evaluation of \text{ArgvList}, for example by the definition of options additional to the standard options known to Xt Toolkit. Each element of \text{OptionList} defines one option, and must be specified in one of the following forms:

\begin{itemize}
  \item \text{[Option, Resource, OptionKind]}
  \item \text{[Option, Resource, noArg, Value]}
\end{itemize}

\begin{align*}
\text{Option} & \quad \text{Atom, an argument in the command line which is being declared.} \\
\text{Resource} & \quad \text{Atom, the name of a resource, the value of which will be defined by this argument.} \\
\text{OptionKind} & \quad \text{Atom, defines how the resource value will be specified. The following may be specified:}
\end{align*}
Predicates

| Predicate          | Definition                                                                 
|--------------------|-----------------------------------------------------------------------------
| noArg              | The value will be specified as the fourth element of this entry, in OptionList. |
| isArg              | The option itself is the value.                                             |
| stickyArg          | The value follows directly after the name of the option.                    |
| sepArg             | The value is the next element of ArgvList.                                  |
| skipArg            | This option and the next element of ArgvList are not to be interpreted by Xt Toolkit. |
| skipLine           | This option and the remainder of ArgvList are not to be interpreted by Xt Toolkit. |

Value Atom, the resource value for options of the noArg type.

InvalidArgumentException is a list of resource settings that will be used in case no resource file is found. Give one atom per resource setting. Otherwise the notation is just like the one used in resource files.

Arguments

WidgetName Atom, the name of a widget
ToolkitName Atom
ApplicationClass Atom
OptionList A list of lists, consisting of three or four atoms: [[ArgvName, ResourceName, ArgKind, Value] ...]
ArgvList A list of atoms
FallbackResources A list of atoms

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

system_error(cannot_open_display) The display which was specified by the environment variable $DISPLAY or in ArgvList cannot be opened.

permission_error(reinitialize) Xt Intrinsics cannot be reinitialized.

Hints

xtInitialize/5 can only be executed repeatedly if it is called with exactly the same parameters.
See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtReInitialize/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtInitialize()</td>
</tr>
</tbody>
</table>
Specify accelerators to operate across windows

\[
\text{xtInstallAccelerators} \ ( +\text{SourceName}, +\text{DestinationName} )
\]

\[
\text{xtInstallAllAccelerators} \ ( +\text{SourceName}, +\text{DestinationName} )
\]

These predicates specify accelerators which operate across windows. Accelerators link user inputs, e.g. pressing a particular sequence of keys, to actions. As soon as installation has been completed using \text{xtInstall...Accelerators/2}, the functions initiated by events will be executed in the context of the destination widget, even if the events occur in the source widget. This mechanism can be used, for example, if it is required to permit menu items to be activated via the keyboard, even though they are normally called up by mouse clicks.

\text{xtInstallAccelerators/2} specifies that the events which are initiated by accelerators in the widget named \text{SourceName} are to be passed to the widget \text{DestinationName}.

\text{xtInstallAllAccelerators/2} specifies that the events which are initiated by accelerators in the widget named \text{SourceName}, or in a widget anywhere in the tree below it, should be passed to the widget \text{DestinationName}.

This applies to all the accelerators which are defined in the current accelerator table for the widget \text{DestinationName}.

Arguments

\begin{itemize}
  \item \text{SourceName} \quad \text{Atom, the name of a widget}
  \item \text{DestinationName} \quad \text{Atom, the name of a widget}
\end{itemize}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

\begin{itemize}
  \item X \quad \text{XtInstallAccelerators()}, \text{...AllAccelerators()}
\end{itemize}
The `xtIsComposite/1` predicate succeeds if the widget named `WidgetName` is an instance of a subclass of the `compositeWidgetClass` type, otherwise it fails.

**Arguments**

- **WidgetItem**
  Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- **X**
  `XtIsComposite()`
Predicates xtIsManaged/1

Check management of a widget

| xtIsManaged ( +WidgetName ) |

The xtIsManaged/1 predicate succeeds if the widget named WidgetName is managed, otherwise it fails.

Arguments

   WidgetName    Atom, the name of a widget

Exceptions

   For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

   X           XtIsManaged()
Check that a widget is realized

xtIsRealized/1

The xtIsRealized/1 predicate succeeds if the widget named \textit{WidgetName} has been realized, otherwise it fails.

Arguments

\begin{itemize}
  \item \texttt{WidgetName} \hspace{1cm} Atom, the name of a widget
\end{itemize}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

\begin{itemize}
  \item X \hspace{1cm} XtIsRealized()
\end{itemize}
Check that a widget can be activated

\[ \texttt{xtIsSensitive( +WidgetName )} \]

The xtIsSensitive/1 predicate checks whether the widget named \textit{WidgetName} can be activated, otherwise it fails.

**Arguments**

- \textit{WidgetName} : Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- \texttt{XtIsSensitive()}
Check class to which a widget belongs

```prolog
# xtIsSubclass ( -WidgetName, -ClassName )
```

xtIsSubclass/2 checks whether the widget named `WidgetName` represents an instance of a subclass of the class specified by `ClassName`.

If `ClassName` is a variable, backtracking can be used to determine all the classes which are superior to the widget’s class.

If `WidgetName` is a variable, backtracking can be used to determine all the widgets which are instances of a particular subclass of the specified class.

If both parameters are variables, backtracking can be used to determine all widget-to-class pairings.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WidgetName</code></td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td><code>ClassName</code></td>
<td>Atom, a defined class name</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

XtIsSubclass()
Predicates

xtKeysymToKeycode/3

Convert key symbol (keysym) to key code (keycode) and vice versa

\[
\text{xtKeysymToKeycode} \ ( +\text{WidgetItem}, +\text{Keysym}, ?\text{Keycode} )
\]

\[
\text{xtKeysymToKeycode} \ ( +\text{WidgetItem}, ?\text{Keysym}, +\text{Keycode} )
\]

xtKeysymToKeycode/3 translates the key symbol \textit{Keysym} into the key code \textit{Keycode} and vice versa. If a value is used for \textit{Keysym} and a variable for \textit{Keycode}, \textit{Keycode} will be returned. If a value is used for \textit{Keycode} and a variable for \textit{Keysym}, \textit{Keysym} will be returned.

Arguments

- **WidgetItem** Atom, the name of a widget
- **Keysym** Atom, a key symbol
- **Keycode** Integer $\geq 0$

See also

- \texttt{XKeysymToKeycode()}, \texttt{XKeysymToString()}
- \texttt{XStringToKeysym()}, \texttt{XKeycodeToKeysym()}
Description

xtKeysymToKeycodeList/4 returns a list of Keycodes that map to a particular Keysym in the keyboard mapping table.

WidgetItem is used to determine the appropriate display.

Keycodes is a list of Keycodes that have Keysym associated with them or the atom 'noSymbol' if KeycodesCount is 0.

KeycodesCount is the number of Keycodes in Keycodes.

Arguments

WidgetItem Atom, the name of a widget
Keysym Atom, a keysym symbol
Keycodes List of keycodes or the Atom 'noSymbol'
KeycodesCount Integer

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtKeysymToKeycode/3
X XtKeysymToKeycodeList()
Predicates xtLoopWhileEvents/0

**Process outstanding events**

<table>
<thead>
<tr>
<th>xtLoopWhileEvents</th>
</tr>
</thead>
</table>

xtLoopWhileEvents/0 executes the event loop until there are no more events to handle. This predicate corresponds to xtMainLoop/0, with the difference that xtLoopWhileEvents/0 is terminated when there are no more events.

X Window events are only processed when the system is executing the event loop. After the creation of a widget, an application can call xtMainLoop/0 or xtLoopWhileEvents/0 to start the user interactions.

xtLoopWhileEvents/0 can be used to permit user actions if it is necessary to perform very long calculations in a callback. For example, a Quit button can be kept active by calling xtLoopWhileEvents/0 from time to time.

xtLoopWhileEvents/0 is not terminated by xtBreakMainLoop/0.

If xtLoopWhileEvents/0 is called by a callback, the jobs for the current event will not be executed fully until xtLoopWhileEvents has terminated.

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtBreakMainLoop/0, xtMainLoop/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtMainLoop()</td>
</tr>
</tbody>
</table>
xtLowerWidget/1 places the window belonging to the widget named `WidgetName` at the bottom of the stack of displayed windows. Together with xtRaiseWidget/1 this predicate provides the only means of affecting the stacking order, for the children of widgets which are not container widgets. For any of these other child widgets, the management of their geometry data by the parents should also include their stacking order.

This predicate can only be executed if the widget `WidgetName` already exists and has been realized.

**Arguments**

- `WidgetName` Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- `domain_error(realized_widget)`
  - The specified widget must have been realized.

**See also**

- IF/Prolog: xtRaiseWidget/1, xtRestackWidget/3
- X: XLowerWindow()
Execute main event loop

xtMainLoop

xtMainLoop/0 executes the main event loop of the application.

X Window events will only be processed when the event loop is executed. After the creation of a widget, an application can call xtMainLoop/0 to start the user interactions.

xtMainLoop/0 waits until an event arrives. It processes these events i.e. the associated callbacks and the actions which are linked to them are called up.

If xtMainLoop/0 is called by a callback, the jobs for the current event will not be executed fully until xtMainLoop has terminated.

xtMainLoop/0 is normally an endless loop, but this can be exited by calling the predicate xtBreakMainLoop/0 within a callback.

See also

IF/Prolog    xtBreakMainLoop/0, xtLoopWhileEvents/0
X            XtMainLoop()
Manage child widgets

xtManageChild ( +WidgetName )

xtManageChildren ( +WidgetList )

These predicates request that the geometry management of a child widget is undertaken by its parent widget.

xtManageChild/1 adds the widget named *WidgetName* to the geometry management of its parent widget.

xtManageChildren/1 can be used to add a list of widgets, specified in *WidgetList*, to the geometry management of their common parent widget.

The parent widget must be a Container widget, if it is to be able to undertake the management of the geometry data for the child widget.

All the widgets in *WidgetList* must have the same parent widget.

Arguments

- **WidgetName** Atom, the name of a widget
- **WidgetList** A list of atoms These atoms are the names of defined widgets

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- **domain_error(parent)** The specified argument must belong to a subclass of compositeWidgetClass.
- **domain_error(unambiguous_parent)** The widgets which are specified must have the same parent widget.

See also

- IF/Prolog xtCreateManagedWidget/5, xtUnmanageChild/1, xtUnmanageChildren/1
- X XtManageChild(), ...ren()
xtMapWidget/1 maps the widget named \textit{WidgetName}. A widget does not become visible on the screen until it has been mapped.

The attribute 'mapped' or 'not mapped' for a widget which does not have a Container widget for its parent can only be changed by using the predicates xtMapWidget/1 and xtUnmapWidget/1, because such widgets are not held in any parent widget’s list of managed child widgets, and hence the mappedWhenManaged resource cannot be used.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom</td>
<td>the name of a widget</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain_error(realized_widget)</td>
<td>The specified widget must have been realized.</td>
</tr>
</tbody>
</table>

**Hints**

Normally, all widgets which are managed by their parent widget will be mapped automatically. This predicate is therefore only seldom required.

**See also**

IF/Prolog : xtManageChild/1, xtSetMappedWhenManaged/1, xtUnmapWidget/1

\textbf{X} : XtMapWidget
Relate widget names to toolkit names

xtName ( +WidgetName, ?ToolkitName )

# xtName ( -WidgetName, +ToolkitName )

xtName/2 retrieves the toolkit name of WidgetName when WidgetName is instantiated or all matching widget names when ToolkitName was supplied.

Arguments

WidgetName Atom, the name of a widget or a variable
ToolkitName Atom or variable

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

IF/Prolog xtWidget/1
X XtName()
Convert widget name to window name

```
# xtNameToWidget ( ?RefWidgetName, +ToolkitNames, ?WidgetName )
```

xtNameToWidget/3 looks in the widget tree specified by RefWidgetName for a widget WidgetName, whose name matches that specified by ToolkitNames.

Here, ToolkitNames is a character string of the form used in the resource files for specifying widgets. The individual components of this character string may be separated by either periods or asterisks.

xtNameToWidget/3 uses the same interpretation mechanism as the resource converter, and returns the widget with the shortest fully qualified name which matches the specified pattern.

If IF/Prolog does not yet know of the widget, a new and unique name will be assigned to it. This name can be used for later references. This is of interest when the user wishes to process complex widgets which consist of child widgets.

If RefWidgetName is not specified, then all the widgets which IF/Prolog knows will be searched for a suitable child or descendant. By backtracking, a number of results can be obtained.

**Arguments**

- **RefWidgetName** Atom, the name of a widget
- **ToolkitNames** Atom, in the form 'ToolkitName1*ToolkitName2...' or 'ToolkitName1.ToolkitName2...'
- **WidgetName** Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- X XtNameToWidget()
xtOverrideTranslations/2

Override Translations

xtOverrideTranslations ( +WidgetName, +Translations )

xtOverrideTranslations/2 is used to override translations of a widget. If a translation of an event already exists it will be replaced. xtOverrideTranslations/2 is syntactically simpler than overriding the translations with xtSetValue/2 (see the example). Translations are usually set in a resource file. You should only hardcode translations, if you don’t want them to be meddled with by a user.

Arguments

- **WidgetName**
  Atom, the name of a widget
- **Translations**
  Atom, event

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Example

The following calls have the same effect:

- `xtOverrideTranslations( widget, 'Ctrl<Key>C: xtActionBye()' )`
- `xtSetValue( widget, [ translations = '
  '#override
  'Ctrl<Key>C: xtActionBye()'
] ).`

See also

- IF/Prolog xtAugmentTranslations/2, xtSetValue/2, xtSetValues/2
- X XtOverrideTranslations()
Find related parent and child widgets

xtParent/2 returns the parent widget $ParentName$ belonging to a child widget $WidgetName$, or vice versa.

If IF/Prolog does not yet know the parent widget, a new unique name will be assigned to this widget. This name can be used for later references.

If $WidgetName$ is a variable, the parents of all the widgets which are known to IF/Prolog will be unified with $ParentName$. By backtracking, all parent-child pairs can be determined.

If a widget has no parents, as will be the case for Shell widgets at the top level, the result will be none (atom).

Arguments

- $WidgetName$: Atom, the name of a widget
- $ParentName$: Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- X: XtParent()
Check for existence of events

**xtPending**

xtPending/0 checks whether there are still events from some event source, which are awaiting processing. The goal will be successful if there are events still outstanding, otherwise it will fail.

This predicate can be used in combination with xtLoopWhileEvents/0 or xtProcessEvent/1.

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtLoopWhileEvents/0, xtProcessEvent/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtPending()</td>
</tr>
</tbody>
</table>
Predicates \hspace{10em} \texttt{xtPopdown/1}

**Terminate output of a pop-up widget**

\[
\texttt{xtPopdown ( +WidgetName )}
\]

\texttt{xtPopdown/1} makes the pop-up widget named \textit{WidgetName} invisible again. Any lock modes which were set are also canceled.

**Arguments**

- \texttt{WidgetName} \hspace{1em} \texttt{Atom}, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Hints**

This action is automatically carried out in the predefined callback predicate, \texttt{xtCallbackPopdown/3}.

**See also**

- IF/Prolog \hspace{1em} \texttt{xtPopup/2, xtAddCallback/4}
- X \hspace{1em} \texttt{XtPopdown()}

Motif Manual 490 IF/Prolog V5.2
Output pop-up widget

xtPopup/2 displays the pop-up widget $WidgetName$; i.e. it is made visible. Pop-ups are widget trees which are visually outside the normal widget hierarchy. They have a separate Shell widget, and hence a separate window hierarchy. They are used to output requests in a dialog, or for similar purposes. The Shell widget is created using the predicate xtCreatePopupShell/5, and must then be made visible by using xtPopup/2.

$GrabKind$ specifies how widgets which lie outside the pop-up are locked for user events.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$WidgetName$</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>$GrabKind$</td>
<td>Atom:</td>
</tr>
<tr>
<td></td>
<td>grabNone</td>
</tr>
<tr>
<td></td>
<td>locks none of the other widgets</td>
</tr>
<tr>
<td></td>
<td>grabNoneExclusive</td>
</tr>
<tr>
<td></td>
<td>locks all the widgets which lie outside a sequence of</td>
</tr>
<tr>
<td></td>
<td>pop-ups</td>
</tr>
<tr>
<td></td>
<td>grabExclusive</td>
</tr>
<tr>
<td></td>
<td>locks all the widgets which lie outside the pop-up</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error($grab\_kind$)

The specification must be one of the atoms: grabNone, grabNoneExclusive or grabExclusive.

Hints

This action is automatically carried out in the predefined callback predicates xtCallbackName/3, xtCallbackExclusive/3 and xtCallbackNonExclusive/3.

See also

IF/Prolog xtPopupSpringLoaded/1, xtPopdown/1, xtAddCallback/4
X XtPopup()
Display pop-up widget

\[\texttt{xtPopupSpringLoaded ( +WidgetName )}\]

xtPopupSpringLoaded/1 displays the pop-up widget \textit{WidgetName}. Pop-ups have a separate Shell widget, and hence a separate window hierarchy. The Shell widget is created using the predicate xtCreatePopupShell/5, and must then be made visible by using xtPopup/2 or xtPopupSpringLoaded/1.

In contrast to xtPopup/2, xtPopupSpringLoaded/1 displays the pop-up widget only as long as the mouse button that popped up the widget is pressed. The widget will be automatically popped down as soon as the mouse button is released. xtPopupSpringLoaded/1 is mainly used to output messages or warnings.

Arguments

- \texttt{WidgetName} Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog \texttt{xtPopup/2}
- X \texttt{XtPopupSpringLoaded()}

Motif Manual 492 IF/Prolog V5.2
xtProcessEvent/1 processes one event, from the event source specified by `EventMask`.

This is only useful if preference is to be given to a particular event source. Normally, events are processed by calling `xtMainLoop/0`. The following are possible event sources:

- `xtIMXEvent` — X server events
- `xtIMTimer` — timer events (see `xtAddTimeout/4`)
- `xtIMAlternateInput` — additional input sources (see `xtAddInput/5``)
- `xtIMAll` — stands for all event sources

`EventMask` is a list of event sources (atoms), which are to be processed.

If `xtProcessEvent/1` is called by a callback, the jobs for the current event will not be executed fully until `xtProcessEvent/1` is completed.

### Arguments

- `EventMask` — A list of atoms: `xtIMXEvent | xtIMTimer | xtIMAlternateInput | xtIMAll`

### Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- `domain_error(input_mask)` — The following list elements may be specified: `xtIMXEvent, xtIMTimer, xtIMAlternateInput, xtIMAll`

### Hints

`xtProcessEvent/1` will wait until an event arrives from the specified event source, if this has not already happened.

### See also

- IF/Prolog: `xtMainLoop/0, xtPending/0`
- X: `XtProcessEvent()`
Predicates

xtRaiseWidget/1

Place widget at top of window stack

xtRaiseWidget( +WidgetName )

xtRaiseWidget/1 puts the window which belongs to the widget named WidgetName at the top of the stack of displayed windows. Together with xtLowerWidget/1 this predicate provides the only means of affecting the stacking order, for the children of widgets which are not container widgets. For any of these other child widgets, the management of their geometry data by the parents should also include their stacking order.

Arguments

WidgetName Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error(realized_widget)

The specified widget must have been realized.

See also

IF/Prolog xtLowerWidget/1, xtRestackWidget/3
X XRaiseWindow()
xtReInitialize/0 releases all the widgets. It is an enhancement to the predicate xtInitialize/5. However, xtInitialize/5 cannot be canceled by xtReInitialize/0, which serves only to release the widgets. Use of xtReInitialize/0 destroys all the widgets except the TopLevelShell widget.

See also

IF/Prolog    xtInitialize/5
Predicates

xtRealizeWidget/1

Realize widget

\[
\text{xtRealizeWidget} \ ( \ +\text{WidgetName} \ )
\]

xtRealizeWidget/1 realizes the widget named \text{WidgetName} and also, if this is a composite widget, the child widgets which appear in its list of managed children.

When a widget is realized, the entire widget tree is output on the screen.

Arguments

- \text{WidgetName} \quad \text{Atom, the name of a widget}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- IF/Prolog \quad \text{xtUnrealizeWidget/1}
- X \quad \text{XtRealizeWidget()}

Motif Manual 496 IF/Prolog V5.2
xtRemoveAllCallbacks/2 Predicates

Remove all the callback predicates in a callback list

xtRemoveAllCallbacks ( +WidgetName, +CallbackName )

XtRemoveAllCallback/2 removes all callback predicates belonging to the callback CallbackName, for the widget named WidgetName.

Arguments

WidgetName Atom, the name of a widget
CallbackName Atom, the name of a callback

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error(widget_callback)
    The specified widget does not have the specified callback list.

See also

IF/Prolog xtAddCallback/4, xtRemoveCallback/4
X XtRemoveCallbacks()
Remove callback predicates

```prolog
xtRemoveCallback ( ?WidgetName, ?CallbackName, ?CallbackPredicate, ?ClientData )
```

```prolog
xtRemoveCallbacks ( ?CallbackList )
```

These predicates remove callback predicates from a callback list.
xtRemoveCallback/4 removes the callback predicate, `CallbackPredicate` for the widget named `WidgetName` from the callback list `CallbackName`. Backtracking is possible for this predicate, as for retract/1/2. If one of the parameters of xtRemoveCallback/4 is specified as a variable, backtracking can be used to remove a number of callbacks.

`ClientData` are the application-specific data for the callback, defined in xtAddCallback/4.
xtRemoveCallbacks/1 calls xtRemoveCallback/4 to remove all the callback predicates from the `CallbackList`.

**Arguments**

- **WidgetName** Atom, the name of a widget
- **CallbackName** Atom, the name of a callback
- **CallbackList** A list of lists These contain the following four elements:
  
  ```prolog
  WidgetName, CallbackName, CallbackPredicate, ClientData
  ```

- **CallbackPredicate** Atom, the functor of a predicate with an arity of three
- **ClientData** Term

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

```prolog
domain_error(callback_name)
```

There is no callback list with the specified name

**See also**

- IF/Prolog `xtAddCallback/4`
- X `XtRemoveCallback(), ...s()`
xtRemoveGrab/1

**Remove input restrictions**

\[
\text{xtRemoveGrab} \ ( \ +\text{WidgetName} \ )
\]

xtRemoveGrab/1 removes the input restrictions for the widget named \textit{WidgetName}. An input restriction can be set by using xtAddGrab/3.

**Arguments**

- \textit{WidgetName} : Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog : xtAddGrab/3
- X : XtRemoveGrab()
xtRemoveInput/1 cancels the input job with the name InputName. InputName is set using xtAddInput/1.

**Arguments**

InputName Atom, the name of an input job

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**existence_error(input_identifier)**

There is no input job set with the specified name

**See also**

IF/Prolog xtAddInput/5

X XtRemoveInput()
xtRemoveTimeout/1 cancels the timeout job with the name \textit{TimeOutName}. \textit{TimeOutName} is set using xtAddTimeOut/4.

**Arguments**

\begin{itemize}
  \item \texttt{TimeOutName} Atom, the name of a timeout job
\end{itemize}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\begin{itemize}
  \item \texttt{existence_error(timeout_identifier)}
  \begin{itemize}
    \item No timeout has been set with the specified name.
  \end{itemize}
\end{itemize}

**See also**

\begin{itemize}
  \item IF/Prolog \texttt{xtAddTimeOut/4}
  \item X \texttt{XtRemoveTimeOut()}
\end{itemize}
Predicates xtRemoveWorkProc/1

**Cancel work job**

xtRemoveWorkProc ( +WorkName )

xtRemoveWorkProc/1 cancels the work job with the name \textit{WorkName}. \textit{WorkName} is set using xtAddWorkProc/3.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorkName</td>
<td>Atom, the name of a work job</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{existence_error(work_identifier)}

There is no work job set with the specified

**See also**

<table>
<thead>
<tr>
<th>Source</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xtAddWorkProc/3</td>
</tr>
<tr>
<td>X</td>
<td>XtRemoveWorkProc()</td>
</tr>
</tbody>
</table>

Motif Manual 502 IF/Prolog V5.2
xtRestackWidget/3 moves the window belonging to \textit{WidgetName} to a higher or lower position in the stack of displayed windows. The new position of the window, relative to its siblings, is specified by means of \textit{StackMode} and \textit{SiblingName}.

The following table shows how \textit{SiblingName} and \textit{StackMode} affect the new position of the window which belongs to \textit{WidgetName}.

<table>
<thead>
<tr>
<th>SiblingName</th>
<th>StackMode</th>
<th>Position of window</th>
</tr>
</thead>
<tbody>
<tr>
<td>above</td>
<td></td>
<td>Directly above the sibling window</td>
</tr>
<tr>
<td>below</td>
<td></td>
<td>Directly below the sibling window</td>
</tr>
<tr>
<td>topIf</td>
<td></td>
<td>At the top of the stack if it is completely or partially covered by the sibling window.</td>
</tr>
<tr>
<td>bottomIf</td>
<td></td>
<td>At the bottom of the stack if it completely or partially covers the sibling window.</td>
</tr>
<tr>
<td>opposite</td>
<td></td>
<td>At the top of the stack if it is completely or partially covered by the sibling window; at the bottom of the stack if it completely or partially covers the sibling window.</td>
</tr>
</tbody>
</table>

**Arguments**

- \textbf{WidgetName}: Atom, the name of a widget
- \textbf{SiblingName}: Atom, the name of a widget
- \textbf{StackMode}: Atom: above | below | topIf | bottomIf | opposite

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \textbf{domain\_error(stack\_mode)}
  
  The stack mode must be specified as one of the atoms: above, below, bottomIf, topIf, or opposite.

- \textbf{domain\_error(sibling)}
  
  The specified argument must be a sibling widget of the specified widget.

- \textbf{domain\_error(realized\_widget)}
  
  The specified widget must have been realized.
Predicates

xtRestackWidget/3

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtLowerWidget/1, xtRaiseWidget/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XConfigureWindow()</td>
</tr>
</tbody>
</table>
Redirect input events

| xtSetKeyboardFocus | ( +SubtreeName, +DescendantName ) |

xtSetKeyboardFocus/2 redirects input events which arrive via the keyboard for the widget `SubtreeName`, to the widget `DescendantName`.

It is possible to specify (the atom) `none` for `DescendantName`. In this case, no widget will be given the keyboard focus.

Arguments

- **SubtreeName**: Atom, the name of a widget
- **DescendantName**: Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

- X `XtSetKeyboardFocus()`
Add internationalisation support

\texttt{xtSetLanguageProc}

\texttt{xtSetLanguageProc/0} should be called directly before the predicate \texttt{xtInitialize/5/6} in order to add internationalization support. \texttt{xtSetLanguageProc/0} calls the C-function \texttt{XtSetLanguageProc(NULL, NULL, NULL)}. Please consult the OSF/Motif reference manuals for further details.

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

Compatibility

\texttt{xtSetLanguageProc/0} is not available in the interface to OSF/Motif 1.1.

See also

\begin{itemize}
\item X \texttt{XtSetLanguageProc()}
\end{itemize}
Output widget

\[
\text{xtSetMappedWhenManaged} \ ( +\text{WidgetName}, \ +\text{Boolean} )
\]

\text{xtSetMappedWhenManaged/2} sets a value for the mappedWhenManaged resource. This resource specifies whether a widget named \text{WidgetName} is to be made visible if it is managed by a parent widget.

**Arguments**

- \text{WidgetName} \quad \text{Atom, the name of a widget}
- \text{Boolean} \quad \text{Atom: true | false | yes | no | on | off}

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog \quad \text{xtMapWidget/1}
- X \quad \text{XtSetMappedWhenManaged()}

Predicates  

xtSetMultiClickTime/2

Set multiclick time

xtSetMultiClickTime ( +WidgetName, +MultiClickTime )

xtGetMultiClickTime/2 determines the time lap in milliseconds that serves to determine whether a multi-click or just a repeated click has occurred.

The translation manager uses this time interval to determine if multiple events are to be interpreted as a repeated event for purposes of matching a translation entry containing a repeat count.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>MultiClickTime</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/Prolog</td>
<td>xtGetMultiClickTime/2</td>
</tr>
<tr>
<td>X</td>
<td>XtSetMultiClickTime()</td>
</tr>
</tbody>
</table>
Set widget so it can be activated

\begin{verbatim}
xtSetSensitive( +WidgetName, +Boolean )
\end{verbatim}

xtSetSensitive/2 sets a value for the \texttt{sensitive} resource to define whether or not the widget named \texttt{WidgetName} can be activated.

\section*{Arguments}

- \texttt{WidgetName} Atom, the name of a widget
- \texttt{Boolean} Atom: true | false | yes | no | on | off

\section*{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\section*{See also}

- \texttt{X} XtSetSensitive()
Predicates xtSetUserSpecifiedLocation/1

Description

xtSetUserSpecifiedLocation ( +ShellWidgetName )

xtSetUserSpecifiedLocation/1 sets the current position of ShellWidgetName’s window as the user specified position hints for the window manager. ShellWidgetName must name a realized shell widget.

Arguments

   WidgetName Atom, the name of a shell widget

Exceptions

   For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

   X XGetWMNormalHints(), XSetWMNormalHints()
Set and modify resources

\[
\text{xtSetValue} ( \ +\text{WidgetName}, +\text{ArgValue} )
\]

\[
\text{xtSetValues} ( \ +\text{WidgetName}, +\text{ArgList} )
\]

These predicates set or modify the values of resources. Some of the resources cannot be modified using these predicates, e.g. pointers to procedures. Resources can also be set in configuration files and, when the application is called, in the command line.

\text{xtSetValue}/2 sets the resource value \text{ArgValue} for the widget named \text{WidgetName}.

\text{xtSetValues}/2 sets the resource values specified in the list \text{ArgList} for the widget named \text{WidgetName}.

The resources associated with the widgets, and their possible values, are described fully in the chapter \textit{Widgets and gadgets} (see page 35).

Arguments

- \text{WidgetName} Atom, the name of a widget
- \text{ArgValue} Value assignment for a resource.
  This value assignment has the form: \text{ResourceName} = \text{ResourceValue} or \text{ResourceName}(\text{ResourceValue}).
- \text{ArgList} A list of value assignments for resources.
  These value assignments have the form: \{\text{ResourceName} = \text{ResourceValue}, ...\} or \{\text{ResourceName}(\text{ResourceValue}), ...\}

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

- \text{domain\_error(resource\_name)} The specified resource identifier does not exist.
- \text{domain\_error(resource\_type)} The specified resource value is not of the required data type.
- \text{domain\_error(resource)} The specified resource is not valid for this widget.
- \text{domain\_error(resource\_list)} A variable may not be specified for a resource value.
- \text{existence\_error(resource\_type)} One of the resources which has been specified cannot be set by IF/Prolog.
permission_error(access)
   This resource cannot be set using xtSetValue.

domain_error(resource)
   The resource value which has been specified is not represented
   in the correct form for this resource.

domain_error(resource_value_pair)
   The list elements which are specified must be value assignments.

Hints

Callbacks can only be specified using xtAddCallback/4.

The IF/Prolog representation of resources is converted to a corresponding representation
in the Toolkit. This is done by a converter provided by the Toolkit, or by a converter
provided by the interface.

See also

IF/Prolog  xtAddCallback/4, xtGetValue(s)/2
X          XtSetValue(), ...s()
Set cursor for a widget

xtSetWidgetCursor ( +WidgetName, +Cursor )

xtSetWidgetCursor/2 changes the cursor appearance for \textit{WidgetName}.

\textbf{Arguments}

- **WidgetName**: Atom, the name of a widget
- **Cursor**: Atom, a cursor name
  Examples: arrow, mouse, pencil

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

- \texttt{X} \quad \texttt{XChangeWindowAttributes()}


Flush output buffer

xtSync/1 flushes the output buffer of the display assigned to the WidgetName widget and waits until all the events and error messages for preceding server calls have been received and executed.

Arguments

WidgetName Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

X XSync()
xtTopLevelShellWidget/1

**Output widget name of the toplevel widget**

| xtTopLevelShellWidget ( ?WidgetName ) |

xtTopLevelShellWidget/1 unifies *WidgetName* with the widget name of the current Top-LevelShell widget of the application.

**Arguments**

- **WidgetName**
  - Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog  
  - xtInitialize/5
Convert coordinates

```
xTranslateCoords ( +WidgetName, +XPosition, +YPosition, ?RootX, ?RootY )
```

`xtTranslateCoords/5` converts an X/Y coordinate pair, `XPosition` and `YPosition`, which are the coordinates of a widget called `WidgetName`, into the root window coordinates `RootX` and `RootY`. This is useful for positioning pop-up widgets.

The window for the widget must already exist, and the widget must have been realized.

**Arguments**

- **WidgetName** Atom, the name of a widget
- **XPosition** Integer
- **YPosition** Integer
- **RootX** Integer
- **RootY** Integer

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

```
domain_error(realized_widget)
```

The specified widget must have been realized.

**See also**

- X `XtTranslateCoords()`
xtUngrabButton/3 Predicate

**Cancel button reservation**

xtUnGrabButton/3 cancels the redirection of button events to *WidgetName*’s window. The cancellation takes place for all matching *Button-Modifiers* combinations used in prior xtGrabButton/9 calls.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Button</td>
<td>Atom: anyButton</td>
</tr>
<tr>
<td>Modifiers</td>
<td>A list of atoms or an empty list. List elements: anyModifier</td>
</tr>
</tbody>
</table>

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**domain_error(modifier)**

The list elements which may be used for the Modifiers argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

**See also**

IF/Prolog       xtGrabButton/9
X               XUngrabButton(), XtUngrabButton()
Predicates

xtUngrabKey/3

**Cancel keyboard reservation by a key**

| xtUngrabKey ( +WidgetName, +Keycode, +Modifiers ) |

xtUnGrabKey/3 cancels the reservation of the keyboard of the display assigned to the WidgetName widget, made when the key Keycode is pressed at the same time as the key defined in Modifiers. Such a reservation is generated by using xtGrabKey/3.

*Modifiers* are one or more additional keys.

**Arguments**

- **WidgetName**: Atom, the name of a widget
- **Keycode**: Integer ≥ 0
  - 0 corresponds to anyKey
- **Modifiers**: A list of atoms or an empty list
  - List elements: anyModifier | shift | control | lock | mod1 | mod2 | mod3 | mod4 | mod5

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**domain_error(modifier)**

- The list elements which may be used for the Modifiers argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

**See also**

- IF/Prolog: xtGrabKey/3
- X: XUngrabKey()
xtUngrabKeyboard/1 cancels the reservation of the keyboard for widget $WidgetName$ for the application.

**Arguments**

- **WidgetName**: Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**See also**

- IF/Prolog: xtGrabKeyboard/4
- X: XUngrabKeyboard()
Predicates xtUngrabPointer/1

**Cancel pointer reservation**

xtUngrabPointer ( +WidgetName )

xtUnGrabPointer/1 cancels the redirection of pointer events to WidgetName’s window.

**Arguments**

WidgetName Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

domain_error(modifier)

The list elements which may be used for the Modifiers argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

**See also**

IF/Prolog xtGrabPointer/7
X XUngrabPointer(), XtUngrabPointer()
Switch off management of a widget

\[\text{xtUnmanageChild} ( +\text{WidgetName} )\]

\[\text{xtUnmanageChildren} ( +\text{WidgetList} )\]

These predicates switch off the management by a parent of the geometry data for a widget, thus making them invisible on the screen. The parent widget is a widget from the CompositeWidgetClass subclass.

\text{xtUnmanageChild/1} switches off management of the widget named \text{WidgetName}.

\text{xtUnmanageChildren/1} has the same effect for all the widgets itemized in the \text{WidgetList}.

**Arguments**

- \text{WidgetName}  
  Atom, the name of a widget
- \text{WidgetList}  
  A list of atoms These atoms are the names of defined widgets

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\text{domain_error(parent)}  
The specified argument must belong to a subclass of compositeWidgetClass.

**Hints**

The predicates \text{xtUnmanageChild/1} and \text{xtUnmanageChildren/1} do not destroy widgets.

**See also**

- IF/Prolog  
  \text{xtManageChild/1, xtManageChildren/1}
- X  
  \text{XtUnmanageChild(), XtUnmanageChildren()}
Cancel mapping of a widget

xtUnmapWidget ( +WidgetName )

xtUnmapWidget/1 cancels the mapping of the widget named *WidgetName*, and it is removed from the screen. A widget is mapped by means of xtMapWidget/1.

This is the only way in which a widget which is not a child of a Composite widget, and all its ”descendants”, can be prevented from being mapped, and hence from being displayed, while at the same time its ancestors are mapped and displayed, i.e. are visible.

Arguments

WidgetName Atom, the name of a widget

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\[
\text{domain\_error(\text{realized\_widget})}
\]

The specified widget must have been realized.

See also

<table>
<thead>
<tr>
<th>IF/Prolog</th>
<th>xtManageChild/1, xtMapWidget/1, xtSetMappedWhenManaged/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XtUnmapWidget()</td>
</tr>
</tbody>
</table>
Delete window for a widget

{xtUnrealizeWidget/1(WidgetItemName,Atom)}

xtUnrealizeWidget/1 deletes all windows that belong to the widget named \textit{WidgetItemName}, including those belonging to its children.

\textbf{Arguments}

WidgetItemName \hspace{1cm} Atom, the name of a widget

\textbf{Exceptions}

For argument type and mode dependent exceptions see section on Exceptions (page 557).

\textbf{See also}

\begin{tabular}{ll}
IF/Prolog & \texttt{xtRealizeWidget/1} \\
X & \texttt{XtUnrealizeWidget()} \\
\end{tabular}
Predicates xtUnregisterWidget/1

**Unregister a widget**

```prolog
xtUnregisterWidget( +WidgetName )
```

xtUnregisterWidget/1 unregisters the widget \textit{WidgetName} from the IF/Prolog management. This predicate should be used with care. One example where it should be used is in Drag-DropFinish callbacks to unregister widgets that get implicitly destroyed by OSF/Motif.

**Arguments**

- `WidgetItem` Atom, the name of a widget

**Exceptions**

For argument type and mode dependent exceptions see section on Exceptions (page 557).

**Compatibility**

xtUnregisterWidget/1 is not available in the interface to OSF/Motif 1.1.
Query the version of X Window System

xtVersion/2

xtVersion (?Version, ?Revision)

xtVersion/2 returns the version number Version and the release number Revision of the X Window System which is being used.

Arguments

Version  Integer, the version of X Window System
Revision  Integer, the release number X Window System

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

X          XtVersion()
Predicate: xtWidget/1, xtWidgetList/1

**Query widget names**

```prolog
# xtWidget ( ?WidgetName )

xtWidgetList ( ?WidgetList )
```

These predicates return the widget names that are known to the IF/Prolog system.

- `xtWidget/1` unifies `WidgetName` with the name of a widget known to IF/Prolog.
- By backtracking it is possible to determine all the widgets which are known to IF/Prolog.
- `xtWidgetList/1` unifies `WidgetList` with a list of all the widget names known to IF/Prolog.

**Arguments**

- `WidgetName` - Atom, the name of a widget
- `WidgetList` - A list of atoms. These atoms are the names of defined widgets.
Query window id or widget name

# xWindowToWidget (?Window, ?WidgetName )

If WidgetName is instantiated, the predicate returns the window id Window of the associated window; or if Window is instantiated, it returns the name WidgetName of the associated widget.

It is sometimes useful during testing to know the window id, for example when X utilities such as xwininfo are being used.

If IF/Prolog does not yet know the widget which belongs to the specified window id, then a new and unique name will be assigned to it. This name can be used for subsequent references. This is of interest when the user wishes to process complex widgets which consist of child widgets.

If neither WidgetName nor Window is instantiated, then by backtracking the system will supply the widget name and window id of every widget which is known to it.

The window id of a widget which is not realized is 0. There is no window with this id.

If there is no matching widget, then WidgetName will be unified with the atom 'none'.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetName</td>
<td>Atom, the name of a widget</td>
</tr>
<tr>
<td>Window</td>
<td>Integer, the window id</td>
</tr>
</tbody>
</table>

Exceptions

For argument type and mode dependent exceptions see section on Exceptions (page 557).

See also

X        XtWindow(), XtWindowToWidget()
Appendix A

Programming examples

The source code for the programs described below is available in the directory PROROOT/DEMOS/motif. Further examples will also be found there, illustrating how to use IF/Prolog for OSF/Motif programming.

A.1 Example of an editor using various OSF/Motif functions

/*
Copyright (C) Siemens Nixdorf Informationssysteme AG, 1991.
Alle Rechte vorbehalten. All rights reserved.
*/

/*
** The following prolog program shows a sample editor demonstrating
** motif functionality such as different widget classes, pulldown menus,
** popup widgets, clipboard functionality.
**
** Each widget will be created in the main program,
** but some of them will not be managed unless they are needed.
**
** Main Widget Hierarchy ( these widgets are permanently visible ) :
**
**
**      toplevel shell
**      |                     |
**      form widget        |
**      |                     |
**      menu bar           | scrolled text
**      |                     |
**      3 cascade buttons  |                     |
** */
The following widgets are not permanently visible:

There are pulldown menus attached to the menue bar’s cascade buttons above:

Furthermore there is a file selection box managed by its own dialog shell, which will be created automatically by the convenience creating function "xmCreateFileSelectionDialog". Via this file selection box you can choose the file you want to edit.

The file selection box inherits behaviour and resources from Core, Composite, Constraint, XmManager, XmBulletin, XmSelectionBox and consists of the following widgets:

- label
- textfield for displaying and editing a directory mask used to select the files to be displayed
- label
- scrolled list for listing filenames
- label
- scrolled list for listing subdirectories
- label
- textfield for displaying and editing a filename
- separator
- 4 push buttons with default labels OK, Filter, Cancel, Help ( in this program labels are changed )

There are two prompt dialog boxes:
One of them will be managed, if the "Save As" button was pressed, for users input of the file name under which the contents of the text widget will be saved.
The other one will be managed, if the "Print" button in the "Config" menu was pressed, for users input of his own print command.
There are two message boxes:
One of them is a "message dialog" which will be managed, when
the help button of the file selection box is activated and
display a help text, the other one is a "warning dialog"
which displays appropriate warnings when an error situation
occurred in an editor session.

** CLIPBOARD:

The second menu offers the motif clipboard facilities for
copying, cutting and pasting text. A clipboard is an invisible
memory for storing data. The data can be exchanged between
applications running on the same display.

For trying out the clipboard facilities you should start this
program two or more times. Cut or copy the primary selection
of the text widget of the one application and then paste the
clipboard’s contents to the text widget of another.

/*
:- import(motif).

program :-
    set_global( print_command, lpr ),
    user_parameters( Params ),

    /*
    ** Initialize toolkit and create form and menubar
    */
    xtInitialize( top, top, 'Prolog_editor', [],
                 [ 'prolog-editor' | Params ] ),
    xtSetValues( top, [ allowShellResize = true ] ),
    xmCreateForm( form, top, form,
                  [ ] ),
    xtManageChild( form ),

    /*
    ** Create menubar
    */
    xmCreateMenuBar( menubar, form, menubar,
                     [ topAttachment = attach_form,
                       [ ] ],

Programming examples

leftAttachment = attach_form,
rightAttachment = attach_form
]
);
xtManageChild( menubar ),

/*
** Create 3 pulldown menu panes and 3 cascade buttons in menubar
*/

xmCreatePulldownMenu( pulldown1, menubar, pulldown1, [] ),
xmCreateCascadeButton( casc1, menubar, casc1,
[ subMenuId = pulldown1,
  labelString = 'Files',
  mnemonic = 'F'
 ] ),
xmCreatePulldownMenu( pulldown2, menubar, pulldown2, [] ),
xmCreateCascadeButton( casc2, menubar, casc2,
[ subMenuId = pulldown2,
  labelString = 'Edit',
  mnemonic = 'E'
 ] ),
xmCreatePulldownMenu( pulldown3, menubar, pulldown3, [] ),
xmCreateCascadeButton( casc3, menubar, casc3,
[ subMenuId = pulldown3,
  labelString = 'Config',
  mnemonic = 'C'
 ] ),
xtManageChildren( [ casc1, casc2, casc3 ] ),

/*
** Create items of the first pulldown menu
*/

xmCreatePushButtonGadget( open_button, pulldown1, open_button,
[ labelString = 'Open',
  mnemonic = 'O'
 ] ),
xtAddCallback( open_button, activateCallback, openCB, file_sel_box ),

xmCreatePushButtonGadget( new_button, pulldown1, new_button,
[ labelString = 'New',
  mnemonic = 'N'
 ] ),

Motif Manual  532  IF/Prolog V5.2
xtAddCallback( new_button, activateCallback, newCB, text ),

xmCreatePushButtonGadget( save_button, pulldown1, save_button,
  [ labelString = 'Save',
    mnemonic = 'S'
  ] ),
xtAddCallback( save_button, activateCallback, saveCB, text ),

xmCreatePushButtonGadget( save_as_button, pulldown1, save_as_button,
  [ labelString = 'Save As',
    mnemonic = 'A'
  ] ),
xtAddCallback( save_as_button, activateCallback, openCB, savedialog ),

xmCreatePushButtonGadget( print_button, pulldown1, print_button,
  [ labelString = 'Print',
    mnemonic = 'P'
  ] ),
xtAddCallback( print_button, activateCallback, printCB, text ),

xmCreatePushButtonGadget( exit_button, pulldown1, exit_button,
  [ labelString = 'Exit',
    mnemonic = 'x'
  ] ),
xtAddCallback( exit_button, activateCallback, exitCB, _ ),

xtManageChildren( [ open_button, new_button, save_button,
          save_as_button, print_button, exit_button ] ),

/*
 ** Create items of the second pulldown menu
 */

xmCreatePushButtonGadget( cut_button, pulldown2, cut_button,
  [ labelString = 'Cut',
    mnemonic = 't'
  ] ),
xtAddCallback( cut_button, activateCallback, cutCB, text ),
xmCreatePushButtonGadget( copy_button, pulldown2, copy_button,
[ labelString = 'Copy', mnemonic = 'C' ] ),
xtAddCallback( copy_button, activateCallback, copyCB, text ),

xmCreatePushButtonGadget( paste_button, pulldown2, paste_button,
[ labelString = 'Paste', mnemonic = 'P' ] ),
xtAddCallback( paste_button, activateCallback, pasteCB, text ),

xtManageChildren( [ cut_button, copy_button, paste_button ] ),

/**************************
** Create items of the third pulldown menu
**
**************************

xmCreatePushButtonGadget( confprint_button, pulldown3, confprint_button,
[ labelString = 'Print', mnemonic = 'P' ] ),
xtAddCallback( confprint_button, activateCallback, confprintCB, printdialog ),

xtManageChildren( [ confprint_button ] ),

/*************************
** Create the text widget for editing files.
** xmCreateScrolledText is a convenience creating function
** which creates a ScrolledWindow widget with a text widget
** and two scrollbars as its children.
**
*************************/

xmCreateScrolledText( text, form, text,
[ rows = 24, columns = 60, topAttachment = attach_widget, topWidget = menubar, leftAttachment = attach_form, rightAttachment = attach_form, bottomAttachment = attach_form, editMode = multi_line_edit ] )
** Create file selection box for selecting the file to edit.

```
xmCreateFileSelectionDialog( file_sel_box, top, file_sel_box, 
[ autoUnmanage = true, 
  okLabelString = 
    [text=open, separator, text=selected, 
      separator, text=file], 
  applyLabelString = 
    [text=change, separator, text=directory] 
  ] ), 
xtAddCallback( file_sel_box, okCallback, okCB, 
[ file_sel_box, text ] ),
xtAddCallback( file_sel_box, cancelCallback, cancelCB, 
  file_sel_box ),
```

** Creating a message box for displaying the help text of the file selection box

```
Message= 
  [Message= [ 
    'Select one file in the file selection box,,'
    'then press button :'
  ]],
xtAddCallback( file_sel_box, helpCallback, helpCB, 
[ mes_box, Message ] ),
list_to_xmString( Message, XmString),
xmCreateMessageDialog( mes_box, top, mes_box, 
[ 
  messageString = XmString 
  ] ),
```

** Create a prompt dialog widget for typing in the desired filename under which the contents of the text widget will be saved.

```
xmCreatePromptDialog( savedialog, top, savedialog, 
[ 
  selectionLabelString = 'Type in <file_name> for saving oat', 
```
okLabelString = 'save',
] ),
xmSelectionBoxGetChild( savedialog, help_button, Help ),
xtUnmanageChild( Help ),
xtAddCallback( savedialog, okCallback, oksaveCB,
[ savedialog, text ] ),
xtAddCallback( savedialog, cancelCallback, cancelCB,
savedialog ),

/**
** Create A warning dialog widget for error messages
*/
xmCreateWarningDialog( warning, top, warning, [] ),
xmMessageBoxGetChild( warning, cancel_button, Cancel_Button ),
xmMessageBoxGetChild( warning, help_button, Help_Button ),
xtUnmanageChildren( [ Cancel_Button, Help_Button ] ),

/**
** Create a prompt dialog widget to configure
** print facility
*/
xmCreatePromptDialog( printdialog, top, printdialog,
[ selectionLabelString = 'Type in the print command !'
] ),
xmSelectionBoxGetChild( printdialog, help_button, PHelp ),
xtUnmanageChild( PHelp ),
xtAddCallback( printdialog, okCallback, okprintCB, printdialog ),
xtAddCallback( printdialog, cancelCallback, cancelCB,
printdialog ),

/**
** Realize the widget tree and enter the event loop
*/
xtRealizeWidget( top ),
xtMainLoop.

/**
** Callback predicate for OK in file selection
*/
okCB( _Widget, [ File_Sel_Box, _TextWidget ], _CallData ) :-
   xtGetValue( File_Sel_Box, textString = XmString ),
xmstring_to_string( XmString, '' ),
messagebox_popup( warning, ["No file selected !"] ), !.
okCB( _Widget, [ File_Sel_Box, TextWidget ], _CallData ) :-
   xtGetValue( File_Sel_Box, textString = XmString ),
xmstring_to_string( XmString, Current_File ),
catch( open( Current_File, read, Stream ), _, fail ),
set_global( current_file, Current_File ),
/*
 ** Read in the file.
 */
get_until( Stream, end_of_file, Text, _ ),
close( Stream ),
xmTextSetString( TextWidget, Text ), !.
okCB( _, _, _ ) :-
    messagebox_popup( warning, ['File not found !'] ).
/*
 ** General callback for popup a dialog widget
 */
openCB( _Widget, Dialog, _CallData ) :-
    xtManageChild( Dialog ).
/*
 ** General callback for popdown a dialog widget
 */
cancelCB( _Widget, Dialog, _CallData ) :-
    xtUnmanageChild( Dialog ).
/*
 ** General callback for help messages
 */
helpCB( _Widget, [ Mess_Box, Message ], _CallData ) :-
    messagebox_popup( Mess_Box, Message ).
/*
 ** Callback for 'New'-button (clear the text)
 */
newCB( _Widget, TextWidget, _CallData ) :-
    unset_global( current_file ),
    xtSetValues( TextWidget, [ value = '', rows = 1,
                                columns = 1 ] ).
/*
 ** Callback for 'Save As' after filename enter
 */
oksaveCB( _Widget, [ Dialog, _Text ], _ ) :-
    xtGetValue( Dialog, textString = XmString ),
    xmstring_to_string( XmString, '' ), !,
    messagebox_popup( warning, 'Empty <file_name> not allowed !' ).
oksaveCB( _Widget, [ Dialog, Text ], _ ) :-
    xtGetValue( Dialog, textString = XmString ),
xmstring_to_string( XmString, SaveFile ),
catch( open( SaveFile, write, Stream ), _, fail ),
write_text( Stream, Text ),
close( Stream ), !.

oksaveCB( _Widget, [ _Dialog, _Text ], _ ) :-
messagebox_popup( warning, 'Could not write file !' ).

={['*/
** Callback for 'Save'-button
*/
saveCB( _Widget, TextWidget, _CallData ) :-
current_global( current_file ),
get_global( current_file, Current_File ),
catch( open( Current_File, write, Stream ), _, fail ), !,
write_text( Stream, TextWidget ),
close( Stream ).

saveCB( _, _, _ ) :-
current_global( current_file ), !,
messagebox_popup( warning, 'Could not write file !' ).

saveCB( _, _, _ ) :-
messagebox_popup( warning, 'Yet no file opened !' ).

=*/
** Callback for 'Print'-button
*/
printCB( _Widget, TextWidget, _CallData ) :-
xmTextGetString( TextWidget, Text ),
Text \= [],
get_global( print_command, Print ),
catch( system( Print, Pipe, user_output ), _, fail ),
system('sleep 5'),
write_text( Pipe, TextWidget ), !,
close( Pipe ).

printCB( _, TextWidget, _ ) :-
xmTextGetString( TextWidget, Text ),
Text \= [], !,
messagebox_popup( warning, 'Printing failed !' ).

printCB( _, _, _ ) :-
messagebox_popup( warning, 'Nothing to print !' ).

=*/
** Callback for 'Exit'-button
*/
exitCB( _Widget, _ClientData, _CallData ) :-
halt( 0 ).
/**
** Cut the selected text and put it
** to the clipboard
*/
cutCB(_Widget, TextWidget, _CallData) :-
catch( xmTextCut( TextWidget ), _, fail ), !.
cutCB(_, _, _) :-
messagebox_popup( warning,
['Nothing to cut !','No text selected !']).

/**
** Put the selected text to the clipboard
*/
copyCB(_Widget, TextWidget, _CallData) :-
catch( xmTextCopy( TextWidget ), _, fail ), !.
copyCB(_, _, _) :-
messagebox_popup( warning,
['Nothing to copy !','No text selected !']).

/**
** Get text from the clipboard an insert it
** at the current insertion position
*/
pasteCB(_Widget, TextWidget, _CallData) :-
catch( xmTextPaste( TextWidget ), _, fail ), !.
pasteCB(_, _, _) :-
messagebox_popup( warning,
['Nothing to paste !','No data available !']).

/**
** Configuration of the printer command
*/
confprintCB(_Widget, PrintDialog, _CallData) :-
get_global( print_command, Command ),
xtSetValue( PrintDialog, textString = Command ),
xtManageChild( PrintDialog ).

okprintCB(_Widget, _PrintDialog, _CallData) :-
xmGetCallbackComponent( valueString = XmString ),
xmstring_to_string( XmString, '' ), !,
messagebox_popup( warning, ['Empty print command !'] ).
okprintCB(_Widget, _PrintDialog, _CallData) :-
xmGetCallbackComponent( valueString = XmString ),
xmstring_to_string( XmString, Command ),
set_global( print_command, Command ).

/*
** This predicate writes the content of the text widget
** to a file.
**
** The text widget stores its value as a list of lines.
** The lines themselves are atoms.
*/
write_text( File, TextWidget ) :-
    xmTextGetString( TextWidget, Text ),
    write_text_list( File, Text ).
write_text_list( _, [] ) :- !.
write_text_list( File, [ LastLine ] ) :-
    write( File, LastLine ), !.
write_text_list( File, [ Line | Rest ] ) :-
    write( File, Line),
    nl( File ),
    write_text_list( File, Rest ).

/*
** This predicate converts a simple XmString to a prolog atom.
*/
xmstring_to_string( XmString, String ) :-
    append( _, [ text = Text | _ ], XmString ), !,
    Text = String.
xmstring_to_string( _, '' ).
list_to_xmString( [], []):-
    !.
list_to_xmString( [First| RestList],
    [text=First, separator| RestXmString]):-
    list_to_xmString( RestList, RestXmString).

/*
** Popup a warning dialog widget and
** show the given message
*/
messagebox_popup( MessageBox, Message ) :-
    list_to_xmString( Message, XmString),
    xtSetValues( MessageBox, [ messageString = XmString ] ),
    xtManageChild( MessageBox ).
A.2 Example showing communication with the Motif Window Manager

/*
 * (c) Copyright 1994 SNI AG
 */

/*
 ** Demonstration program on the use of the protocol predicates
 ** for communication with the Motif Window Manager.
 **
 ** In this, a number of items are added into MWM’s Default
 ** Window Menu, and the program uses appropriate MWM protocols
 ** to inform it when one of these items has been clicked on,
 ** so that it can then start the corresponding actions.
 */

:- import( motif ).

program :-
    user_parameters( Params ),
    xtInitialize( Top, Top, 'Protocol', [], [ protocol | Params ]),
    xtCreateManagedWidget( Draw, Draw, xmDrawWidgetClass, Top,
                         [ width = 300, height = 300 ]),
    changeWMmenu( Top, Draw ),
    xtRealizeWidget( Top ),
    xtMainLoop.

program :-
    exception( 'Motif WM not running' ).

/*
 ** Create the entries in the MWM menu, and register the
 ** protocol callback predicates.
 */

changeWMmenu( Top, Draw ) :-
    xmIsMotifWMRunning( Top ),
    xmInternAtom( draw_draw, false, Xdraw_draw ),
    xmInternAtom( draw_clear, false, Xdraw_clear ),
    xmInternAtom( draw_exit, false, Xdraw_exit ),
    concat_atom( [ 'Draw', 'f.send_msg', Xdraw_draw ], ' ', DrawEntry ),
    concat_atom( [ 'Clear', 'f.send_msg', Xdraw_clear ], ' ', ClearEntry ),
    concat_atom( [ 'Exit', 'f.send_msg', Xdraw_exit ], ' ', ExitEntry ),
    concat_atom( [ DrawEntry, ClearEntry, ExitEntry ], '\n', Menu ),
    xtSetValue( Top, mwmMenu = Menu ),
xmAddProtocolCallback( Top, '_MOTIF_WM_MESSAGES', draw_draw, draw_draw, Draw ),
xmAddProtocolCallback( Top, '_MOTIF_WM_MESSAGES', draw_clear, draw_clear, Draw ),
xmAddProtocolCallback( Top, '_MOTIF_WM_MESSAGES', draw_exit, draw_exit, Draw ).

/*

** Protocol callback predicates for implementing the
** appropriate functions.
*/

draw_draw( _, Draw, _ ) :-
    xtDrawFillArc( Draw, 10, 10, 280, 280, 0, 360 * 64).

draw_clear( _, Draw, _ ) :-
    xtDrawClear( Draw ).

draw_exit( _, _, _ ) :-
    xtCallbackBye( _, _, _ ).
A.3 Pop-up menu example

/*
** (c) Copyright 1994 SNI AG
*/

/*******************************************************************
**  Simple example of usage of pop-up menus and submenus  
**  
*******************************************************************/

:- import( motif ).

program :-
    user_parameters(Parms),
    /*
    ** Initialize Toolkit
    */
    xtInitialize(toplevel, toplevel, '', [], [popup_submenu| Parms]),
    xtAddAction(post_it),
    /*
    ** Create RowColumn in toplevel with two PushButtons
    */
    xmCreateRowColumn(row_col, toplevel, row_col, [
        background = yellow,
        translations = '#augment <Btn3Down>: post_it(popup)'
    ]),
    xtManageChild(row_col),
    xmCreatePushButton(button1, row_col, button1, []),
    xtAddCallback(button1, activateCallback, buttonCB, '1'),
    xmCreatePushButton(button2, row_col, button2, []),
    xtAddCallback(button2, activateCallback, buttonCB, '2'),
    xtManageChildren([button1, button2]),
    /*
    ** Create pop-up menu
    */
    xmCreatePopupMenu(popup, row_col, popup, []),
    /*
    ** Create two submenus and CascadeButtons in the pop-up menu
    */
    xmCreatePulldownMenu(submenu1, popup, submenu1, []),
    xmCreateCascadeButtonGadget(casc1, popup, casc1, [}
subMenuId = submenu1,
labelString = 'First Submenu'
),
xmCreatePulldownMenu(submenu2, popup, submenu2, []),
xmCreateCascadeButtonGadget(casc2, popup, casc2, [
    subMenuId = submenu2,
    labelString = 'Second Submenu'
]),
xtManageChildren([casc1, casc2]),
/*
** Create two PushButtonGadgets in submenu1
*/
xmCreatePushButtonGadget(button1a, submenu1, button1a, []),
xtAddCallback(button1a, activateCallback, buttonCB, '1a'),
xmCreatePushButtonGadget(button1b, submenu1, button1b, []),
xtAddCallback(button1b, activateCallback, buttonCB, '1b'),
xtManageChildren([button1a, button1b]),
/*
** Create two PushButtonGadgets in submenu2
*/
xmCreatePushButtonGadget(button2a, submenu2, button2a, []),
xtAddCallback(button2a, activateCallback, buttonCB, '2a'),
xmCreatePushButtonGadget(button2b, submenu2, button2b, []),
xtAddCallback(button2b, activateCallback, buttonCB, '2b'),
xtManageChildren([button2a, button2b]),
/*
** Create a submenu of submenu1
*/
xmCreatePulldownMenu(submenu3, submenu1, submenu3, []),
xmCreateCascadeButtonGadget(casc3, submenu1, casc3, [
    subMenuId = submenu3,
    labelString = 'To Third Submenu'
]),
xtManageChildren([casc3]),
/*
** Create pushbuttons in submenu3
*/
xmCreatePushButtonGadget(button3a, submenu3, button3a, []),
xtAddCallback(button3a, activateCallback, buttonCB, '3a'),
xmCreatePushButtonGadget(button3b, submenu3, button3b, []),
xtAddCallback(button3b, activateCallback, buttonCB, '3b'),
xtManageChildren([button3a, button3b]),
/*
** Realize widgets and start event loop
*/
*/
   xtRealizeWidget(toplevel),
   xtMainLoop.

post_it(Widget, params(Menu)) :-
   xmMenuPosition(Menu),
   xtManageChild(Menu).

buttonCB(Widget, ClientData, CallData) :-
   write('Button '), write(ClientData), write(' selected'), nl.
## Appendix B

### Callback structure components

<table>
<thead>
<tr>
<th>Class</th>
<th>Reasons</th>
<th>Components</th>
<th>IF/Prolog type</th>
</tr>
</thead>
<tbody>
<tr>
<td>all callbacks</td>
<td>all</td>
<td>reason</td>
<td>atom: *1</td>
</tr>
<tr>
<td>xmArrowButton</td>
<td>all</td>
<td>click_count</td>
<td>integer</td>
</tr>
<tr>
<td>xmArrowButtonGadget</td>
<td>all</td>
<td>click_count</td>
<td>integer</td>
</tr>
<tr>
<td>xmCommand</td>
<td>all</td>
<td>valueString</td>
<td>list</td>
</tr>
<tr>
<td>xmDragContext</td>
<td>all</td>
<td>timeStamp</td>
<td>opaque</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drop_finish</td>
<td></td>
<td>completionStatus</td>
<td>drop_success, drop_failure</td>
</tr>
<tr>
<td>drop_start</td>
<td></td>
<td>dropAction</td>
<td>atom: drop, drop_cancel, drop_help, drop_interrupt, drop_interupt, drop_suspended</td>
</tr>
<tr>
<td>drop_finish</td>
<td></td>
<td>dropAction</td>
<td></td>
</tr>
<tr>
<td>drop_site_enter</td>
<td></td>
<td>operation</td>
<td>atom: drop_move, drop_copy, drop_link, drop_noop</td>
</tr>
<tr>
<td>drag_motion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drop_start</td>
<td></td>
<td>operation</td>
<td></td>
</tr>
<tr>
<td>drop_finish</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operation_changed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drop_site_enter</td>
<td></td>
<td>operations</td>
<td>list: drop_move, drop_copy, drop_link, drop_noop</td>
</tr>
<tr>
<td>drag_motion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drop_start</td>
<td></td>
<td>operations</td>
<td></td>
</tr>
<tr>
<td>drop_finish</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operation_changed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>top_level_enter</td>
<td></td>
<td>drag_protocol_style</td>
<td>atom drag.drop_only, drag.dynamic, drag_none, drag_work, drag_idle, drag_get, drag_put, drag_notify, drag_ack</td>
</tr>
</tbody>
</table>

547
<table>
<thead>
<tr>
<th>Class</th>
<th>Reasons</th>
<th>Components</th>
<th>IF/Prolog type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>drag_start</td>
<td></td>
<td>drag_preregister</td>
</tr>
<tr>
<td></td>
<td>drag_motion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>drop_finish</td>
<td>dropSiteStatus</td>
<td>drop_site_valid,</td>
</tr>
<tr>
<td></td>
<td>operation_changed</td>
<td></td>
<td>drop_site_invalid,</td>
</tr>
<tr>
<td></td>
<td>drop_site_enter</td>
<td></td>
<td>no_drop_site</td>
</tr>
<tr>
<td></td>
<td>drag_motion</td>
<td>x</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>drop_site_enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>drop_start</td>
<td>y</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>drop_site_enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>value</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>window</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>window</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>click_count</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>operation</td>
<td>drop_move,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_copy,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_link,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_noop</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>operations</td>
<td>drop_move,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_copy,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_link,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_noop</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>x</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>y</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>dropSiteStatus</td>
<td>drop_site_valid,</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td></td>
<td>drop_site_invalid</td>
</tr>
<tr>
<td></td>
<td>all but drop_message</td>
<td>animate</td>
<td>atom: true, false</td>
</tr>
<tr>
<td></td>
<td>drop_message</td>
<td>dropAction</td>
<td>atom: drop_help</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>drop_help</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stringValue</td>
<td>list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mask</td>
<td>list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dir</td>
<td>list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pattern</td>
<td>list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>item</td>
<td>list</td>
</tr>
</tbody>
</table>
### Callback structure components

<table>
<thead>
<tr>
<th>Class</th>
<th>Reasons</th>
<th>Components</th>
<th>IF/Prolog type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>item_position</td>
<td>integer</td>
</tr>
<tr>
<td>multiple_select</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>browse_select</td>
<td></td>
<td>selected_item_positions</td>
<td>list:</td>
</tr>
<tr>
<td>extended_select</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multiple_select</td>
<td></td>
<td>selected_items</td>
<td>list:</td>
</tr>
<tr>
<td>browse_select</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extended_select</td>
<td></td>
<td>selected_item_count</td>
<td>integer</td>
</tr>
<tr>
<td>extended_select</td>
<td></td>
<td>selection_type</td>
<td>atom: initial, modification, addition,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmPushButton</td>
<td>all</td>
<td>click_count</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmPushButtonGadget</td>
<td>all</td>
<td>click_count</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmRowColumn</td>
<td>activate</td>
<td>widget</td>
<td>atom, widget</td>
</tr>
<tr>
<td></td>
<td></td>
<td>callbackstruct</td>
<td>list: *2</td>
</tr>
<tr>
<td>xmScale</td>
<td>all</td>
<td>value</td>
<td>integer</td>
</tr>
<tr>
<td>xmScrollBar</td>
<td>all</td>
<td>value</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>to_top</td>
<td>pixel</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>to_bottom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmSelectionBox</td>
<td>all</td>
<td>valueString</td>
<td>list</td>
</tr>
<tr>
<td>xmText</td>
<td>losing_focus</td>
<td>doit</td>
<td>atom: true, false</td>
</tr>
<tr>
<td></td>
<td>modifying_text_value</td>
<td>currInsert</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>moving_insert_cursor</td>
<td>newInsert</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>losing_focus</td>
<td>startPos</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>modifying_text_value</td>
<td>endPos</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>modifying_text_value</td>
<td>text</td>
<td>atom</td>
</tr>
<tr>
<td>xmTextField</td>
<td>analogous to xmText</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmToggleButton</td>
<td>all</td>
<td>set</td>
<td>atom: true, false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmToggleButton-Gadget</td>
<td>all</td>
<td>set</td>
<td>atom: true, false</td>
</tr>
</tbody>
</table>

*1 reason is one of the atoms:
activate, help
apply, increment
arm, input
browse_select, losing_focus
cancel, lose_primary
cascading, map
clipboard_data_delete, modifying_text_value
clipboard_data_request, moving_insert_cursor
command_changed, multiple_select
command_entered, no_match
decrement, none
<table>
<thead>
<tr>
<th>Callback Structure Components</th>
<th>Appendix</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_action</td>
<td>ok</td>
</tr>
<tr>
<td>disarm</td>
<td>page_decrement</td>
</tr>
<tr>
<td>drag</td>
<td>page_increment</td>
</tr>
<tr>
<td>draw_ascii</td>
<td>resize</td>
</tr>
<tr>
<td>draw_button</td>
<td>single_select</td>
</tr>
<tr>
<td>draw_motion</td>
<td>to_bottom</td>
</tr>
<tr>
<td>execute</td>
<td>to_top</td>
</tr>
<tr>
<td>expose</td>
<td>unknown</td>
</tr>
<tr>
<td>extended_select</td>
<td>unmap</td>
</tr>
<tr>
<td>focus</td>
<td>value_changed</td>
</tr>
<tr>
<td>gain_primary</td>
<td></td>
</tr>
<tr>
<td>drag_drop_finish</td>
<td>drag_motion</td>
</tr>
<tr>
<td>drop_finish</td>
<td>drop_site_enter</td>
</tr>
<tr>
<td>drop_site_leave</td>
<td>drop_start</td>
</tr>
<tr>
<td>operation_changed</td>
<td>top_level_enter</td>
</tr>
<tr>
<td>top_level_leave</td>
<td>drop_site_enter_message</td>
</tr>
<tr>
<td>drop_site_leave_message</td>
<td>drag_motion_message</td>
</tr>
<tr>
<td>operation_changed_message</td>
<td>drop_message</td>
</tr>
</tbody>
</table>

*2 The callback structure components specified as arguments in a list will be returned.

In OSF/Motif 1.2 the reason components of Drag & Drop callbacks do not have unique (numerical) values in at least some implementations. Thus a list of (possible) reasons will be returned in these cases.
## Event structure components

<table>
<thead>
<tr>
<th>Type</th>
<th>Components</th>
<th>IF/Prolog type</th>
</tr>
</thead>
<tbody>
<tr>
<td>all types</td>
<td>eventType</td>
<td>atom:*1</td>
</tr>
<tr>
<td></td>
<td>eventSendEvent</td>
<td>atom: true, false</td>
</tr>
<tr>
<td></td>
<td>eventEventWindow</td>
<td>integer, window id</td>
</tr>
<tr>
<td>buttonPress</td>
<td>eventRoot</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>eventSubwindow</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>eventX</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventY</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventXRoot</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventYRoot</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventState</td>
<td>list: *2</td>
</tr>
<tr>
<td></td>
<td>eventButton</td>
<td>atom: *3</td>
</tr>
<tr>
<td>circulateNotify</td>
<td>eventWindow</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>eventPlace</td>
<td>atom: placeOnTop, placeOnBottom</td>
</tr>
<tr>
<td>circulateRequest</td>
<td>eventParent</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>eventPlace</td>
<td>atom: placeOnTop, placeOnBottom</td>
</tr>
<tr>
<td>clientMessage</td>
<td>eventMessageType</td>
<td>atom</td>
</tr>
<tr>
<td></td>
<td>eventFormat</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventData</td>
<td>atom or list: *9</td>
</tr>
<tr>
<td></td>
<td>eventDataMsg</td>
<td>atom: *10</td>
</tr>
<tr>
<td></td>
<td>eventDataAtom</td>
<td>list: *11</td>
</tr>
<tr>
<td>colormapNotify</td>
<td>eventColormap</td>
<td>integer, colormap id</td>
</tr>
<tr>
<td></td>
<td>eventColormapState</td>
<td>atom: colormapInstalled, colormapUninstalled</td>
</tr>
<tr>
<td></td>
<td>eventNew</td>
<td>atom: true, false</td>
</tr>
<tr>
<td>configureNotify</td>
<td>eventX</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventY</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventWidth</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventHeight</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventBorderWidth</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventOverrideRedirect</td>
<td>atom: true, false</td>
</tr>
<tr>
<td></td>
<td>eventAbove</td>
<td>integer, window id</td>
</tr>
<tr>
<td>configureRequest</td>
<td>eventWindow</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>eventX</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventY</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventWidth</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventHeight</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>eventParent</td>
<td>integer, window id</td>
</tr>
<tr>
<td></td>
<td>eventBorderWidth</td>
<td>integer</td>
</tr>
</tbody>
</table>

---

IF/Prolog V5.2 551 Motif Manual
Event structure components

- **configureNotify**
  - eventAbove: integer, window id
  - eventWindow: integer, window id
  - eventX: integer
  - eventY: integer
  - eventWidth: integer
  - eventHeight: integer
  - eventParent: integer, window id
  - eventBorderWidth: integer
  - eventOverrideRedirect: atom: true, false
  - eventWindow: integer, window id

- **createNotify**
  - eventWindow: integer, window id
  - eventX: integer
  - eventY: integer
  - eventWidth: integer
  - eventHeight: integer
  - eventParent: integer, window id
  - eventOverrideRedirect: atom: true, false

- **destroyNotify**
  - eventWindow: integer, window id

- **enterNotify**
  - eventRoot: integer, window id
  - eventSubwindow: integer, window id
  - eventX: integer
  - eventY: integer
  - eventXRoot: integer
  - eventYRoot: integer
  - eventState: list: *2
  - eventMode: atom: *5
  - eventInOutDetail: atom: *6

- **leaveNotify**
  - eventRoot: integer, window id
  - eventSubwindow: integer, window id
  - eventX: integer
  - eventY: integer
  - eventXRoot: integer
  - eventYRoot: integer
  - eventState: list: *2
  - eventMode: atom: *5
  - eventInOutDetail: atom: *6

- **focusIn**
  - eventMode: atom: *5
  - eventInOutDetail: atom: *6

- **focusOut**
  - eventMode: atom: *5
  - eventInOutDetail: atom: *6

- **graphicsExpose**
  - eventX: integer
  - eventY: integer
  - eventWidth: integer
  - eventHeight: integer

- **gravityNotify**
  - eventX: integer
  - eventY: integer
  - eventWindow: integer, window id

- **keyPress**
  - eventRoot: integer, window id

- **keyRelease**
  - eventWindow: integer, window id
  - eventSubwindow: integer, window id
  - eventX: integer
  - eventY: integer
  - eventXRoot: integer
  - eventYRoot: integer
### Appendix

<table>
<thead>
<tr>
<th>Event structure components</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventState list: *2</td>
</tr>
<tr>
<td>eventKeycode list: *8</td>
</tr>
<tr>
<td>leaveNotify (see enterNotify)</td>
</tr>
<tr>
<td>mapNotify eventOverrideRedirect atom: true, false</td>
</tr>
<tr>
<td>eventWindow integer, window id</td>
</tr>
<tr>
<td>mappingNotify eventCount integer</td>
</tr>
<tr>
<td>eventRequest atom: mappingModifier, mappingKeyboard, mappingPointer</td>
</tr>
<tr>
<td>eventFirstKeycode list: *8</td>
</tr>
<tr>
<td>mapRequest eventParent integer, window id</td>
</tr>
<tr>
<td>eventWindow integer, window id</td>
</tr>
<tr>
<td>motionNotify eventRoot integer, window id</td>
</tr>
<tr>
<td>eventSubwindow integer, window id</td>
</tr>
<tr>
<td>eventX integer</td>
</tr>
<tr>
<td>eventY integer</td>
</tr>
<tr>
<td>eventXRoot integer</td>
</tr>
<tr>
<td>eventYRoot integer</td>
</tr>
<tr>
<td>eventState list: *2</td>
</tr>
<tr>
<td>propertyNotify eventAtom atom</td>
</tr>
<tr>
<td>eventPropertyState atom: propertyNewValue, propertyDelete</td>
</tr>
<tr>
<td>reparentNotify eventX integer</td>
</tr>
<tr>
<td>eventY integer</td>
</tr>
<tr>
<td>eventParent integer, window id</td>
</tr>
<tr>
<td>eventOverrideRedirect atom: true, false</td>
</tr>
<tr>
<td>eventWindow integer, window id</td>
</tr>
<tr>
<td>resizeRequest eventWidth integer</td>
</tr>
<tr>
<td>eventHeight integer</td>
</tr>
<tr>
<td>selectionClear eventSelection atom</td>
</tr>
<tr>
<td>selectionNotify eventSelection atom</td>
</tr>
<tr>
<td>eventTarget atom</td>
</tr>
<tr>
<td>eventProperty atom</td>
</tr>
<tr>
<td>eventRequestor integer, window id</td>
</tr>
<tr>
<td>selectionRequest eventSelection atom</td>
</tr>
<tr>
<td>eventTarget atom</td>
</tr>
<tr>
<td>eventProperty atom</td>
</tr>
<tr>
<td>eventOwner integer, window id</td>
</tr>
<tr>
<td>unmapNotify eventWindow integer, window id</td>
</tr>
<tr>
<td>eventFromConfigure atom: true, false</td>
</tr>
<tr>
<td>visibilityNotify eventVisibilityState atom: *7</td>
</tr>
</tbody>
</table>

*1 eventType is one of the following atoms:
noEvent
buttonPress
buttonRelease
circulateNotify
circulateRequest
clientMessage
colormapNotify
configureNotify
configureRequest
createNotify
destroyNotify
derectorNotify
enterNotify
expose
focusIn
focusOut
graphicsExpose
gravityNotify
keyPress
keyRelease
keymapNotify
leaveNotify
mapNotify
mapRequest
mappingNotify
motionNotify
noExpose
propertyNotify
reparentNotify
resizeRequest
selectionClear
selectionNotify
selectionRequest
unmapNotify
visibilityNotify

*2 eventState can contain the following atoms as list elements:

button ... button5
shift
control
lock
mod ... mod5

*3 eventButton is one of the following atoms:

button ... button5

*4 eventStackDetail is one of the following atoms:

above
below
topIf
bottomIf
opposite

*5 eventMode is one of the following atoms:

notifyGrab
notifyNormal
notifyUngrab
notifyWhileGrabbed

*6 eventInOutDetail is one of the following atoms:

notifyAncestor
notifyDetailNone
notifyNonlinearVirtual
notifyPointer
### Event structure components

- notifyInferior
- notifyNonlinear
- notifyPointerRoot
- notifyVirtual

*7 eventVisibilityState is one of the following atoms:

- visibilityFullyObscured
- visibilityPartiallyObscured
- visibilityUnobscured

*8 eventKeycode is a three-element list containing the ASCII character string associated with the key (cf. XLookupString()) as an atom, the keycode as an integer, and the key symbol (cf. XKeysymToString()) as an atom.

*9 eventData is interpreted in accordance with eventFormat:

```
eventFormat 8 = atom
16 = list of integers
32 = list of 5 floating point numbers
```

*10 eventDataMsg is interpreted as though the event came from xmSendClientMessage/5. The message sent from there is delivered as an atom.

*11 eventDataAtom is interpreted as a list of X atoms. These atoms (except for NULL entries) are supplied as a list.
Appendix C

Exceptions

The ISO standard for prolog defines the way exceptions are to be handled for IF/Prolog predicates that are part of the ISO set of predicates.

Predicates in the IF/Prolog interface to OSF/Motif handle exceptions in exactly the same way. Exceptions pertaining to the argument type and call mode of the predicates are listed below. Exceptions that pertain to special OSF/Motif requirements are also listed with the predicates.

**domain_error**(requested\_domain)

The argument is of the requested type, but it doesn’t lean to certain domain requirements. (requested\_domain may be positive integer, not\_less\_than\_zero, ...\_widget, ...)

**environment_error**(error)

An operation in the IF/Prolog environment (operating system, OSF/Motif) could not be (succesfully) performed.

**existence_error**(object\_type)

The object on which an operation is to be performed does not exist.

**instantiation_error**

The argument is an input argument(+). The argument must not be a variable, but a variable was specified.

**permission_error**(operation)

It is not permitted to perform an operation at this point.

**system_error**(error)

An error that could not be classified elsewhere has occurred.

**type_error**(requested\_type)

The argument must be of the requested type, if not a variable, but is a term of another type. requested\_type may be atom, integer, ...
Exceptions in alphabetical order

Only the type exceptions that are generated by the predicates of the Motif interface are listed here. For general syntax errors and type exceptions please refer to the appendix to the IF/Prolog Reference Manual [1].

- **domain_error(XmFontList)**
  The specified argument must be an identifier for a font list.

- **domain_error(alignment)**
  The specified argument must be one of the atoms alignment_center, alignment_beginning or alignment_end.

- **domain_error(arcMode)**
  The specified argument must be one of the atoms arcChord or arcPieSlice.

- **domain_error(bitmapX)**
  The specified argument must be one of the atoms bitmap0, ... bitmap9.

- **domain_error(callback_component)**
  The specified callback component is not known to the interface.

- **domain_error(callback_component)**
  The specified callback component is not known to the interface.

- **domain_error(callback_name)**
  There is no callback list with the specified name

- **domain_error(capStyle)**
  The specified argument must be one of the atoms capNotLast, capButt, capRound or capProjecting.

- **domain_error(cascadeButton_widget_or_gadget)**
  The specified argument must be a widget or a gadget of the CascadeButton type.

- **domain_error(childspecifier)**
  Depending on the widget type, the specified argument must be one of the following atoms:
  - Command-Widget:
    command_text ⎜ prompt_label ⎜ history_list
  - FileSelectionBox-Widget:
    apply_button ⎜ cancel_button ⎜ default_button ⎜ help_button ⎜ filter_label ⎜ filter_text ⎜ list ⎜ list_label ⎜ ok_button ⎜ selection_label ⎜ text
  - Message-Box-Widget:
    cancel_button ⎜ default_button ⎜ help_button ⎜ message_label ⎜ ok_button ⎜ separator ⎜ symbol_label
  - SelectionBox-Widget:
    apply_button ⎜ cancel_button ⎜ default_button ⎜ help_button ⎜ list ⎜ list_label ⎜ ok_button ⎜ selection_label ⎜ text ⎜ separator ⎜ work_area
Appendix

Exceptions

domain_error(class_component)
The specified component is not allowed for this callback structure, it is not an element of the callback structure of the calling widget.

domain_error(clipboard_bad_format)
The format could not be registered, e.g. because it already exists.

domain_error(clipboard_data)
The specified argument contains non-convertible data (e.g. uninstantiated variables).

domain_error(colorX)
The specified argument must be one of the atoms color0, ... color9.

domain_error(command_widget)
The specified widget must be of the Command type.

domain_error(component_list)
The elements specified in the list must be value assignments.

domain_error(coordMode)
The specified argument must be one of the atoms: coordModeOrigin or coordModePrevious.

domain_error(direction)
The specified argument must be one of the atoms: string_direction_l_to_r or string_direction_r_to_l.

domain_error(directionSpecifier)
The specified argument must be one of the atoms: traverse_current, traverse_down, traverse_home, traverse_left, traverse_next, traverse_next_tab_group, traverse_prev, traverse_prev_tab_group, traverse_right or traverse_up.

domain_error(draw_widget)
The specified widget must be of the Draw type.

domain_error(equal_list_lengths)
The specified argument must be a list of the same length as a list in another argument of this predicate.

domain_error(event_component)
The structure component which has been specified is not known to the Motif interface.

domain_error(fileSelectionBox_widget)
The specified widget must be of the FileSelectionBox type.

domain_error(file_format)
The specified file does not have the expected contents.

domain_error(fillRule)
The specified argument must be one of the atoms evenOddRule or windingRule.
domain_error(fillStyle)
The specified argument must be one of the atoms fillSolid, fillTiled, fillStippled or fillOpaqueStippled.

domain_error(fontX)
The specified argument must be one of the atoms font0, ... font9.

domain_error(font_list_entry)
The specified argument must be a font list entry structure of the form: entry(name, type, tag).

domain_error(font_list_type)
The specified argument must be a type identifier for a font list entry: is_font or is_font_set.

domain_error(gcX)
The specified argument must be one of the atoms gc0, ... gc9.

domain_error(grab_kind)
The specification must be one of the atoms: grabNone, grabNoneExclusive or grabExclusive.

domain_error(input_mask)
One of the atoms xtInputReadmask, xtInputWriteMask or xtInputExceptMask must be specified.

domain_error(input_mask)
The following list elements may be specified: xtIMXEvent, xtIMTimer, xtIMAAlternateInput, xtIMAAll.

domain_error(integer)
The specified argument must be an integer.

domain_error(internal_class_code)
The specified argument must be the name of a valid widget class.

domain_error(internal_hierarchy_id)
An invalid value was specified for the HierarchyId argument.

domain_error(joinStyle)
The specified argument must be one of the atoms joinMiter, joinRound or joinBevel.

domain_error(keyboard_mode)
The specified argument must be one of the atoms grabModeSync or grabModeAsync.

domain_error(lineStyle)
The specified argument must be one of the atoms lineSolid, lineOnOffDash or lineDoubleDash.

domain_error(list)
The specified list does not have the required structure.
domain_error(list)
    The specified argument must be a list.

domain_error(list_length)
    The specified list is too long.

domain_error(list_or_atom)
    The specified argument must be an atom or a list.

domain_error(list_widget)
    The specified widget must be of the List type.

domain_error(main_widget)
    The specified widget must be of the MainWindow type.

domain_error(messageBox_widget)
    The specified widget must be of the MessageBox type.

domain_error(modifier)
    The list elements which may be used for the Modifiers argument are: anyModifier, shift, control, lock, mod1, mod2, mod3, mod4, mod5

domain_error(no_gadget)
    This predicate cannot be used for gadgets.

domain_error(no_gadget)
    For the widget which has been specified there is no callback list with the specified name.

domain_error(optionlist)
    The specified list contains illegal elements.

domain_error(orientation)
    The specified argument must be one of the atoms horizontal or vertical.

domain_error(parameter_type)
    An invalid value has been specified for the Which parameter.

domain_error(parent)
    The specified argument must belong to a subclass of compositeWidgetClass.

domain_error(pixel_resource)
    The value specified for the foreground or background is not permitted. Conversion to the X Toolkit representation will fail.

domain_error(pointer_mode)
    The specified argument must be one of the atoms grabModeSync or grabModeAsync.

domain_error(positive_integer)
    The argument must be a positive integer (≥ 0).
domain_error(primitive_or_manager_widget)
The specified argument must be the name of either a Primitive or a Manager widget.

domain_error(realized_widget)
The specified widget must have been realized.

domain_error(reason_component)
The specified component is not allowed for this callback structure.

domain_error(rectangle)
The specified argument must be either the atom none or a rectangle specification.

domain_error(rectangle_or_null)
The specified argument must be either the empty list or a list of the coordinates and side lengths for rectangles.

domain_error(resource)
The resource value which has been specified is not represented in the correct form for this resource.

domain_error(resource)
The specified resource is not valid for this widget.

domain_error(resource_list)
A variable may not be specified for a resource value.

domain_error(resource_name)
The specified resource identifier does not exist.

domain_error(resource_type)
The specified resource value is not of the required data type.

domain_error(resource_value_pair)
The list elements which are specified must be value assignments.

domain_error(resource_value_pair)
The specified list elements must be value assignments.

domain_error(resource_value_pair)
The resources must be specified as value assignments.

domain_error(rowColumn_widget)
The specified widget must be of the RowColumn type.

domain_error(scale_widget)
The specified widget must be of the Scale type.

domain_error(scrollBar_widget)
The specified widget must be of the ScrollBar type.
domain_error(scrolledWindow_widget)
    The specified widget must be of the ScrolledWindow type.

domain_error(selectionBox_widget)
    The specified widget must be of the SelectionBox type.

domain_error(shapeHint)
    The specified argument must be one of the atoms complex, nonconvex or convex.

domain_error(shell_widget)
    The specified argument must be the name of a Shell widget.

domain_error(sibling)
    The specified argument must be a sibling widget of the specified widget.

domain_error(stack_mode)
    The stack mode must be specified as one of the atoms: above or below.

domain_error(stack_mode)
    The stack mode must be specified as one of the atoms: above, below, bottomIf, topIf, or opposite.

domain_error(subwindowMode)
    The specified argument must be one of the atoms: clipByChildren or includeInferiors.

domain_error(tag_type)
    The specified TagType is not permitted.

domain_error(text_field_widget)
    The specified widget must be of the TextField type.

domain_error(text_widget)
    The specified widget must be of the Text type.

domain_error(toggleButton_widget)
    The specified argument must be a widget of the ToggleButton type.

domain_error(type_component)
    The specified structure component is not permitted for this event structure.

domain_error(unambiguous_parent)
    The widgets which are specified must have the same parent widget.

domain_error(unit_type)
    The specified arguments are not permissible units of measure.

domain_error(unused_widget_name)
    The specified widget name is already in use.
domain_error(visibility_expected)
The specified argument must be one of the atoms visibility_unobscured, visibility_partially_obscured or visibility_fully_obscured.

domain_error(widget)
The specified argument must be a defined widget name.

domain_error(widget_callback)
The specified widget does not have the specified callback list.

domain_error(xmStringTable_value)
The specified argument must be a list of valid XmStrings.

domain_error(xmString_value)
The specified argument does not have the correct representation for an XmString.

environment_error(clipboard_fail)
It was not possible to successfully execute the predicate.

environment_error(clipboard_locked)
The clipboard has been locked by another application.

environment_error(clipboard_no_data)
The clipboard contains no data elements which correspond to the requested format.

environment_error(could_not_open_file)
The specified file could not be opened for writing.

environment_error(dump_failed)
The predicate could not be successfully executed.

environment_error(file_not_found)
The specified file cannot be opened for reading.

environment_error(file_not_found)
One of the specified files could not be found.

environment_error(mrm_function_failed)
It was not possible to successfully execute the MRM function concerned.

environment_error(mrm_partial_success)
It was possible to execute the predicate partially. (Motif 1.1 only)

environment_error(illegal_format_length)
The only values which may be specified for the argument are 8, 12 or 32.

environment_error(text_function_failed)
There is no text in the clipboard

environment_error(too_many_procedures_registered)
A maximum of 30 MRM callback procedures is permitted.
existence_error(input_identifier)
   There is no input job set with the specified name

existence_error(resource_type)
   One of the specified resources cannot be set by IF/Prolog.

existence_error(timeout_identifier)
   No timeout has been set with the specified name.

existence_error(work_identifier)
   There is no work job set with the specified

permission_error(access)
   This resource cannot be set using xtSetValue.

permission_error(create,input_predicate)
   There is already an input with the specified name.

permission_error(create, timeout_predicate)
   There is already a timeout job with the name specified.

representation_error(exceeded_max_actions)
   The maximum number of global actions which can be defined is 10

representation_error(too_many_arguments)
   No more than 100 elements are permitted in UILArgList.

system_error(callback_predicate_failed)
   A callback predicate may not fail.

system_error(cannot_open_display)
   The display which was specified by the environment variable $DISPLAY or in ArgvList
cannot be opened.

system_error(input_predicate_failed)
   An input predicate which has been activated has not terminated successfully. (The
   exception is not raised until the input predicate is executed).

system_error(no_callback_struct_to_analyze)
   The predicate xmGetCallbackComponent(s)/1 may only be called from within a callback.

system_error(no_event_to_analyze)
   The xtGetEventComponent(s)/1 predicate may only be called from within a callback.

system_error(timeout_predicate_failed)
   A timeout predicate which has been activated has not terminated successfully. (The
   exception is not raised until the timeout predicate is executed).

type_error(clipboard_data_type)
   The Data argument must have the data type specified in the argument FormatName.
Appendix D

Glossary

accelerator key

The user may define a key combination to be used in selecting a particular menu item. Instead of opening a window in a menu to select the item, it is then possible to press the key combination, thus effecting a more rapid access to the item.

access control list

To each server the user can assign a list of computers, the users of which will be allowed to access the server after it has been started. In addition to the users of these computers, all the users of the local computer may access the local servers. The list referred to is held in the file /etc/Xdisplaynumber .hosts. displaynumber is the number of the display on the local computer which is managed by the server. When the server is started for a particular display, it reads the appropriate file, and sets up a copy of this. The user can access this copy using the client xhost.

active window

An active window is a window which has the input focus for the keyboard.

ancestor

An ancestor of window A is another window, B, to which window A is assigned. A is the descendant of its ancestor B. If A is directly assigned to B, then B is called the parent of A. On the other hand, if A is assigned to window C, and this in turn is assigned to window B, then C is the parent of A, C and B are ancestors of A.

A window can only be seen within its ancestors. Apart from the root window, each window has exactly one parent. The root window is the ancestor of all the windows for a screen.
background

The background of a window is a single color or a pattern, depending on whether the user specifies it as a color or a pixmap. The foreground text or graphics will be displayed against this background, in the color or pattern specified for the foreground.

backing_store

backing_store is a storage area in the server.

The server can manage such a storage area for any window, for use in saving its contents when it is obscured by the actions of other windows. When the window which was covered is exposed, the server is then able to rebuild the window contents without reference to the client. Not every server will support this function all the time.

bitmap

A bitmap is a pixmap with a depth of 1.

bit plane

The k-th bits for every pixel in a pixmap form the k-th bit plane of the pixmap. The number of bit planes per window depends on the number of bits per pixel - the depth of the pixmap. A screen is capable of displaying all the bit planes if it provides at least as many color levels as there are bit planes.

button

There are 2 types of buttons:

- The keys on the mouse, also referred to as mouse buttons.
- The rectangular areas in windows, onto which the user can click to request that a function be executed; these are also referred to as window buttons.

button binding

A button binding is a binding of a mouse button to an mwm function. This binding allows the user to specify what has to be done with the button to initiate the mwm function, i.e. pressing it, releasing it or, for example, dragging the mouse with the button held down. In addition to such dependencies on a single button, it is also possible to specify modifier keys which have to be pressed simultaneously with the main key in order to execute the mwm function.
**button event**

A button event is an event which is sent from the server to a client, to inform the latter that a button on the mouse has been pressed or released in a particular window, or that the mouse has been moved with a button held down.

**character**

The alphanumerical characters, special characters and graphics elements of a character set, without any specification of how they are displayed in terms of their width and height, stroke width or other quantitative sizes, are referred to as the characters. On the other hand, the full display specification of a character is called a glyph.

In general, the term font will be used as a synonym for character set, and glyph for character. However, a distinction must be made between the two terms in those contexts in which the displayed form of a character is meant.

**character code**

A character code is the code which the computer uses internally to identify a character.

**character set**

A character set is a collection of alphanumerical characters or graphics elements (e.g. for the mouse cursor), held under a name, the name of the character set. The alphanumerical characters, special characters and graphics elements of a character set, without any specification of how they are displayed in terms of their width and height, stroke width or other quantitative sizes, are referred to as the characters. On the other hand, the full display specification of a character is called a glyph.

In general, the term font will be used as a synonym for character set, and glyph for character. However, a distinction must be made between the two terms in those contexts in which the displayed form of a character is meant.

**child**

A child widget of widget A is another widget, B, which is directly assigned to widget A. Widget A is the parent of widget B. A widget may have a number of child widgets (children). On the screen, all the children of a common ancestor lie on top of each other in a particular order, referred to as the stacking order. All the children of widget A, and their children, grandchildren, etc., are called the descendants of widget A.
client

A client is an application program which exchanges data with a server on the same computer or a different one. To do so, the client established a link with the server, for example a TCP link. A server handles all the links independently of each other, even where there are a number to the same client.

client area

The client area is a rectangular area in a client window.

If a client requests a new window, and if this is to be managed by MWM, then MWM will request a new window from the server. This window is subdivided into a border zone, the MWM window frame and decoration, and a rectangular area within the window, the client area.

The MWM window frame enables the user to change its size, position and other such attributes. The client area is the area in which the client makes its outputs. In general, the client area has an inner border, which enforces a predefined gap between the border of the client area and the text or graphics it contains. A distinction should be made between this inner border of the client area and the matte, and also the MWM window frame, both the latter being managed by the window manager, MWM.

client window

There are some clients which require no window. However, in general a client will expect inputs and outputs via a window. For this purpose, it will make requests to the server to open windows. Each window that a client opens for input or output is a client window of this client. The client windows which are children of the root window are called primary windows, all other windows are secondary windows.

colormap

The colormap for a window defines the colors which are available for the window which is linked to this colormap. It defines the screen color to which each value of a pixel should be converted. The colormaps which are supported by a server will depend on the visual class selected.

A colormap has rows and columns. Each column corresponds to one of the three colors red, green and blue. Each line is a color entry and produces a particular color on the screen. The appearance of each color entry, and how it is accessed, is determined by the visual class.

Depending on the hardware, it is possible to define a number of colormaps. Irrespective of the hardware restrictions, each client can define any number of virtual colormaps. The window manager can load these virtual colormaps. If the storage space provided by the
hardware for colormaps is fully occupied, then loading a further colormap will cause one which has already been loaded to be deleted, provided that the screen hardware permits this.

**colormap focus**

The colormap focus specifies which colormap is currently being used by the server. The colormap focus is on the colormap assigned to the window which is the active window, i.e. which has the input focus. If a number of colormaps are defined, the client may also explicitly set the focus on a particular colormap.

**descendant**

A descendant of widget A is another widget, B, which is assigned to widget A. A is then also referred to as an ancestor of B. If widget A is the direct ancestor of B, then B is the child of A. On the other hand, if widget B is assigned to widget C, and this in turn is assigned to widget A, then B is the child of C, and C and B are descendants of A.

A widget can only be seen within its ancestors. Apart from the root window, each window is a descendant of the root window.

**display**

A display consists of one or more screens, plus a keyboard and a mouse. Since each server manages exactly one display, the term display is often used as a synonym for the server. Accordingly, the environment variable DISPLAY contains all the information, namely the computer name, display number and screen number, required to address a server uniquely across the entire network.

The term mouse is used here as a generic name for an input device which can be used to indicate very quickly to the computer a particular position on the screen. Another example of such a device is the graphics tablet. The current position of the mouse is marked on the screen by the mouse cursor, or mouse pointer. By contrast, the text cursor marks the position in a window at which keyboard inputs will be inserted. Both the mouse cursor and the text cursor can have a different appearance, depending on their positions on the screen.

The mouse has buttons, which can be used to select objects in appropriate positions, items in menus, for example. The keyboard, on the other hand, has keys.

**event**

An event is a report from the server to a client about an event, normally transmitted asynchronously. An event provides information about actions using the mouse (button events) or keyboard (key events), and about the window which had the input focus at the time the
event occurred. Each client determines for itself which events it wants to process, and will also only be provided by the server with the events which it has explicitly requested. The client can request the server to pass on events to other clients.

font

A font is a collection of alphanumeric characters or graphics elements (e.g. for the mouse cursor), held under a name, the font name. The alphanumeric characters, special characters and graphics elements in a font are called glyphs. A glyph is the displayed form of a character. In particular, the width and height, stroke width and other quantitative sizes of a glyph are specified. A font also specifies the spacing of lines and glyphs.

In general, the term font will be used as a synonym for character set, and glyph for character. However, a distinction must be made between the two terms in those contexts in which the displayed form of a character is meant.

The client xlsfonts lists all the fonts which are available to the current display.

In OSF/Motif, a font is always made available by the server. Clients can request particular fonts from the server, but must allow for the possibility that the server cannot access this font. When the server receives a request for a font, it will attempt to load the font from one of the directories specified for the server in the font path. Further details about the font path will be found in the description of the server.

foreground

The foreground of a window is a single color or a pattern, depending on whether the user specifies it as a color or a pixmap. The foreground text or graphics will be displayed against the background of the window in the color or pattern specified for the foreground.

glyph

A glyph is the displayed form of a character. It is an element of a font, namely an alphanumeric character, a special character or a graphics element. By contrast with a character, the width and height, stroke width and other metric sizes of a glyph are specified.

In general, the term font will be used as a synonym for character set, and glyph for character. However, a distinction must be made between the two terms in those contexts in which the displayed form of a character is meant.

gray out

"Gray out" means "identify as temporarily unusable".

In the default OSF/Motif configuration, any parts of a window which are normally accessible to a user, but which cannot in the present situation be accessed, will be grayed out. An
example is items in a menu which would not initiate a meaningful action in the current situation.

**icon**

An icon is a small window which represents another window but which takes up less space on the screen than the actual window.

Under MWM, an icon consists of an icon image and an icon label.

**icon box**

The icon box is a window in which icons can be packed together. When use is made of the MWM icon box, a window may exist simultaneously in two states: as an icon in the icon box, and as a normal window.

**input focus**

The input focus for the keyboard can be assigned to a window. The window then has the input focus, which means that the client which manages this window will receive from the server all information concerning inputs made via the keyboard.

A window which has the input focus for the keyboard is said to be active.

**key**

For each server, each key on the keyboard is assigned an unalterable key code. Any desired key symbol may be assigned to this key code using the client xmodmap. It is this key symbol that the client interprets as an input. The key code can be modified using the modifier keys. Any key code can be assigned other key symbols, depending on which of these modifier keys is held down at the same time. The assignment of modifier keys to key symbols can also be defined by means of the xmodmap client.

**key binding**

A key binding is the binding of a key to an mwm function. In this binding, it is possible to specify additional modifier keys, which must be pressed together with the individual key in order to execute the mwm function.

**key event**

A key event is an event which is sent from the server to a client to inform the latter that a particular key has been pressed in a particular window, and any modifier keys which were held down at the same time.
matte

The matte is part of the window decoration. It lies between the client area of a client window and the size change border, which is also part of the window decoration. The user can configure the matte using the resources.

menu

A menu is a window which contains a list of menu items. It is possible to click onto a menu item with the mouse, and thereby request execution of the function which is defined for this menu item.

mnemonic

The mnemonic for a menu item is a single letter of the alphabet, which appears somewhere in the menu item. It uniquely identifies this menu item within the entire menu. In the menu item, this letter is underlined on the display. A mnemonic allows the user to select the corresponding menu item, when the menu is open, by pressing the key.

parent

Direct ancestor. The parent of window A is another window, B, to which window A is directly assigned. If A is a child of B, then B is called the parent of A. On the other hand, if window A is assigned to window C, which is in turn assigned to window B, then C is the parent of A, and C and B are ancestors of A.

A window can only be seen within its ancestors. Apart from the root window, each window has exactly one parent. The root window is the ancestor of all the windows for a screen.

plane

The k-th bits for every pixel on a screen form the k-th plane of the screen. The maximum number of planes for a screen depends on the hardware. The number of planes for a screen is one of the factors determining the color display possibilities (including gray scales) on the screen.

Pixmap

A pixmap is a two-dimensional array of pixels, the smallest element on a screen which it is possible to address. In a pixmap, the number of bits is the same, say n, for each pixel. The pixmap is then said to have a depth of n, or to have n bit planes. These can be displayed by a screen provided that it has at least n planes. If there are n bit planes, there are 2^n colors available to the user for each pixel.
Typical uses of pixmaps are for icons, the mouse pointer, and the background or foreground pattern.

**primary window**

Primary windows are those client windows which are children of the root window. All other client windows are called secondary windows.

**resource**

Resources permit the user to customize the user interface. Using the resources, it is possible to influence the decoration, size, position, color and pattern of windows, the shadowing to produce a 3D effect, typefaces in windows, and the processing of inputs.

**root window**

Each screen in a display has one root window. All other windows for this screen are descendants of the root window.

**save under**

*save under* is a storage area in the server.

The server can manage a storage area of this type, for use in saving the contents of any window when it is obscured by another window. When the window which was covered is exposed, the server is then able to repaint the window contents without reference to the client. Not every server will support this functionality all the time.

**scroll area**

The scroll area is that part of the inputs and outputs from a client which can be seen in a window. A client can store more than this visible part in an area called the scroll buffer. Some clients can be configured so that the contents of the scroll buffer can be inspected using a scroll bar.

**scroll bar**

A scroll bar is a widget which enables the user to lay the scroll area over any desired part of the scroll buffer. For example, in windows which contain text there is often a vertical scroll bar. A scroll bar consists of a long thin rectangle, at the ends of which are scroll arrows which the user can click onto to move the scroll area in the corresponding direction, and a slider. The position of the slider along the rectangle indicates the position of the scroll area relative to the complete scroll buffer.
secondary window

Secondary windows are all those client windows which are not primary windows, i.e. which are not children of the root window.

server

A server is a program which manages a display and all the links between this display and clients. Since each server manages exactly one display, the term display is often used as a synonym for the server. A server handles all the links independently of each other, even where there are a number to the same client. The server supplies the window mechanism, distributes graphics requests from clients to the correct screens in the correct windows, and in the opposite direction it supplies the inputs to the client which is managing the window which is currently active.

stacking order

Children with the same parent are arranged in a particular order in the window stack, namely from top to bottom. The top window is fully visible. It will cover the others, either completely or partially. The order within the window stack can be changed by explicitly putting a window in the top position or the bottom position in the stacking order.

3D appearance

In general, OSF/Motif displays visible objects on the screen in three-dimensional form, if the objects are managed by MWM. To achieve this 3D display in OSF/Motif, any of the resources provided by the MWM for shadowing an object (for example the resource instance bottomShadowColor for the foreground class) can be used.

transient window

A transient window is a client window which, depending on the situation, is the only one of all the client’s windows, or may be the only one out of all the windows for the screen, which will currently accept any input. Since it will inhibit inputs from what may be a large group of windows, such a window normally has a short lifetime. Menus are examples of transient windows.

visual class

The visual class for a window or an icon specifies the color display capabilities. Whether or not these can be realized will depend on the screen. There are 6 visual classes: StaticGray, StaticColor, TrueColor, GrayScale, PseudoColor, DirectColor. The visual classes are
arranged hierarchically: 'StaticGray' is a derivative of 'GrayScale'; 'GrayScale' and 'StaticColor' are derivatives of 'PseudoColor'; 'PseudoColor' and 'TrueColor' are derivatives of 'DirectColor'.

In the GrayScale class, the values for red, green and blue are equal in every colormap, so that overall a gray value is produced. The gray values can be changed.

The GrayScale visual class can be regarded as a degenerate subclass of PseudoColor, because in PseudoColor the values for red, green and blue may differ from each other, by contrast with GrayScale.

The PseudoColor visual class can be regarded as a degenerate subclass of DirectColor because in contrast to PseudoColor, not only can the values for red, green and blue differ from each other in DirectColor, but also for each pixel the first part (which specifies the value for red) can be used independently of the second part (the value for green) and from the third part (for blue); in the case of PseudoColor, a combination of the values for red, green and blue is assigned to the complete pixel.

StaticGray is the static variant of GrayScale, StaticColor the static variant of PseudoColor, and TrueColor the static variant of DirectColor. In the case of the static variants, the entries in the colormap cannot be changed.

**widget**

A widget is a window which has built-in functionality; for example, a scroll bar is a widget.

**window**

A window is a rectangular region on a screen. It has window attributes, which define the background of the window, its appearance, size and position. The user can assign a value to many of the window attributes, but not to all of them, by means of resources; the user interface can thus be configured to meet individual requirements.

A window is always the child of another window, unless it is the root window for the screen. The windows thus form a hierarchy of windows. Among other things, this hierarchy determines what can be seen of the window and how events are processed.

In general, a window will be managed by two clients: the client which created the window, and the window manager. The window manager manages the window attributes, while the client manages the client area.

**window decoration**

The window decoration for any window determines the functionality of its MWM window frame, as provided for this window by mwm. The user can define the associated functions using an mwm resource. The window decoration includes the resize border, the matte, the
maximize button, the window menu button, the minimize button, and the title area of the window.

The maximize button, minimize button and resize border can be used to change the size of the window. The window menu button is used to post the window menu; this is done by clicking the button with the mouse. The matte makes the client window clearer, and the title area displays the name of the window.

window manager

The window manager is a client which manages children of the root window for a display. It enables the user to give windows a different size or position on the screen, to alter the stacking order, to turn windows into icons and back again, and performs other similar functions. However, the window manager has no effect on the client area of a window. The client is the sole manager of this client area.

Depending on the layout guidelines, which determine the behavior of the window manager, the window manager will also change or render ineffective values specified by clients for the layout of their windows.
Bibliography

[3] IF/Prolog V5.2 OSF/Motif Interface
[4] IF/Prolog V5.2 Informix Interface
[5] IF/Prolog V5.2 Constraints Package
[6] IF/Prolog V5.2 Quick Reference
[7] IF/Prolog V5.2 Windows Interfaces
[8] IF/Prolog V5.2 BDD Package


Index

drawing a filled rectangle, 417

accelerator
  operating across windows, 469
accelerators, 10
action, 9
  call, 397
  determine cause, 446
Action predicate
  adding, 383
action table, 9
application
  retrieve name and class, 449
ApplicationShell, 37
ArrowButton gadget, 40
ArrowButton widget, 38
atom
  querying name, 272

basic elements
drawing, 171
bitmap
  querying size, 408
BulletinBoard, 41
button
  canceling a reservation, 514
  reserving, 458
button event
  redirecting, 458

callback, 9
  checking existence of, 465
callback list, 9, 385
callback predicate, 385
  adding, 209
  deleting, 495
  removing, 318
callback predicates
  deleting, 494
callback structure
  retrieving a component, 273
callback structures, 14
CascadeButton
  highlighting, 215
CascadeButton gadget, 47
CascadeButton widget, 45
character string
  converting, 248
  converting to units, 211
  outputting, 421, 438, 442, 443
  querying the size, 439
child widget
  creating, 400
  managing, 400
child widgets
  managing, 480
circular arcs
  drawing, 406
class membership
  checking, 470
client, 6
clipboard
  cancel data copying, 216
  canceling data copying operations, 238
  delayed copying of data, 219, 221
  ending copying of data, 224, 225
  format name for data, 227
  length of data, 228
  locking, 230
  pending copy operations, 229
  preparing for data copying, 233
  registering format names, 231
  retrieving data, 232, 235
  start copying data to, 217
  unlocking, 237
clipboard store
  deleting a data element, 236
Index

clipboard, number of formats, 226
color value
  fetching, 195
Command, 48
  appending text to the command, 239
  outputting an error message, 240
  overwrite command in list, 242
  querying a component, 241
Composite, 50
compound string
  underlined, 444
compound strings, 13
Constraint, 51
Container widget, 7
converting units, 243
coordinates
  converting, 513
Core, 52
cursor
  select widget, 378
  set for widget, 510
cut and paste, 341, 359
destination address, 275
data types, 16
DialogShell, 55
directory
  searching for a, 265
Display, 56
display, 6
  outputting height, 405
  outputting width, 405
Display widget, 7
DragContext, 58
DragIcon, 63
Draw, 65, 169
  basic elements, 171
  basic graphics elements, 169
  clearing screen contents, 409
  clearing the queue for output operations, 418
  clearing window area, 410
  copying a bitmap, 412
  copying a window area, 411
  deleting a line, 413
  drawing a filled polygon, 416
drawing filled circular arcs, 414
drawing filled elliptical arcs, 414
drawing lines, 422
drawing points, 423
drawing rectangles, 428
drawings, options, 169
  origin for pattern, 436
  outputting an error message, 420
  querying the size of a character string, 438
  querying the size of a font, 419
  reading memory dump, 426
  recalling the graphics context, 427
  repainting the screen contents, 429
  rubber-banding, 435
  rubber-banding mode, 440
  setting the graphics context, 432
Draw, actions, 69
Draw, repainting window contents, 68
Draw, saving operations, 67
Draw, saving the graphics context, 437
drawing functions, 169
DrawingArea, 70
DrawnButton, 72
DropSite, 74
event
  processing, 490
  retrieving a component, 452
event handling, 13
event loop
  executing, 479
event, processing, 21
events
  checking existence, 486
expose event, 13
expose events
  executing, 379
file
  searching for a, 265
FileSelectionBox, 77
  retrieving components, 264
font list, 17
font lists, 13, 14
font size
  setting, 337
Form, 80
Frame, 85
Gadget, 87
gadget
  ArrowButton, 40
  CascadeButton, 47
  Label, 93
gadget, PushButton, 116
gadget, Separator, 144
gadget, ToggleButton, 157
gadgets, 9
graphics context, 170, 171
  active, 171
  changing, 173
  modifying, 172
  saving, 172
  setting, 432
graphics context store, 170, 171
Graphics context, saving, 437
icon value
  converting to pixmap, 196
IF/Prolog, differences from C, 12
input events
  redirecting, 502
input job
  canceling, 497
input predicate
  setting, 389
input restrictions
  removing, 496
keyboard
  cancel reservation, 516
  canceling a reservation, 515
  reserving, 460–462
keyboard event
  redirecting, 460, 461
  keyboard event, processing mode, 317
  keycode
    converting, 475
  keysym
    converting, 475, 476
Label widget, 90
line pattern
  setting, 431
line segments
  drawing, 430
lines
  drawing, 422
List, 94
list
  cancel markings on entries, 294
  checking existence of an entry, 298
  checking if list entry is selected, 300
  deleting entries, 292
  horizontal scroll bar, 307
  inserting entries, 290
  marking an entry, 304
  querying bounding box of an item, 301
  querying position, 311
  querying position of a list item, 295
  querying position of an entry, 296
  querying positions of marked entries, 297
  querying the position of an entry, 299
  replacing entries, 302
  setting location cursor, 309
  setting the last visible entry, 306
  specifying first visible entry, 308
  updating list of selected items, 310
list, setting add mode, 305
LoopWhileEvents, 477
main event loop, 21
main loop
  interrupting, 395
MainWindow, 99
  separator, 312
  specifying children, 313
Manager, 101
mapping a widget, 504
memory dump
Index

creating, 441
reading, 426

menu
querying the environment, 281
setting the pointer, 338

menu cursor, 278

menu item
button, 316
label, 316

MenuShell, 104
MessageBox, 105
component, 315

Motif Resource Manager, 29, 204
converting icon value to pixmap, 196
fetching a color value, 195
initializing, 203

Motif Window Manager, 7
closing the hierarchy, 194
Motif window manager, 6
Motif Window Manager
check status, 288

motion event, 66

motion event, compressing, 66

MRM, 29
assignment of UIL names and values, 205
assignment of UIL names to callback predicates, 206
closing the hierarchy, 194
converting icon value to pixmap, 196
creating hierarchy id, 204
fetch color value, 195
initializing, 203
opening files, 204

multiclick
retrieve time, 454
set time, 505

mwm, 6
status check, 288

notational conventions, 2

OSF/Motif, 7
OSF/Motif, Style Guide, 7
OSF/Motif
querying the version, 380

outlining
rubber-banding mode, 173
output buffer
flushing, 445, 511
OverrideShell, 107

PanedWindow, 108
pattern
setting the origin, 436
pixel, 19

pixmap, 19
point
drawing, 423

pointer
canceling a reservation, 517
querying its position, 420
querying position of, 425
reserving, 463

pointer event
redirecting, 463

pop-up
terminating output, 487

pop-up shell
creating, 402

pop-up widget
outputting, 488, 489

position pop-up menu, 314

Primitive, 111
protocol
deactivating, 249
removing, 319

protocol facilities
testing, 335

Protocol Manager, 319
defining protocol, 210

Protocol Manger

passing protocols to, 214
removing protocols, 322

protocols
activating, 208

PushButton gadget, 116
PushButton widget, 114

rectangles
drawing, 428

resource
modifying value of, 508
setting value of, 508
resource database
  retrieve information, 450
Resource Manager, 8
resource values
  setting equal, 200
resource, database, 8
resource, value of, 8
resources, 8
  retrieve values of, 455
  retrieving values, 447
resources, modifying, 23
resources, setting, 23
restricting input, 388
RowColumn, 117
rubber-banding mode, 173
  resetting, 440
  setting, 435
Scale, 126
scale
  position of the slider, 330
Screen, 129
screen
  retrieve color information, 451
screen contents
  clearing, 409
  repainting, 429
scroll bar
  horizontal, 307
ScrollBar, 132
  position of the slider, 331
ScrolledWindow, 136
  components, 333
SelectionBox, 139
  querying a component, 334
Separator gadget, 144
Separator widget, 143
server, 6
Shell, 145
shortcut selection, accelerator, 10
syntax
  notational conventions, 2

removing a widget, 320
Text, 147
  canceling a marking, 340
  canceling input restrictions, 346
  check position of a character, 375
  checking editability, 349
  clipboard, 341, 359
  copying, 341, 359
  deleting text, 361
  inserting text, 358
  maximum length of character string, 352
  moving text, 363
  outputting text, 374
  outputting text marking, 370
  position of first baseline, 347
  position of first character, 357, 373
  position of insertion cursor, 350, 368
  position of marked text, 354
  position of the last character, 351
  querying a character string, 355
  querying input restrictions, 348
  querying marked text, 353
  querying position of a character, 360
  replacing text, 362
  setting add mode, 364
  setting editability, 366
  setting highlighting, 367
  setting maximum length of character string, 369
  specifying a source, 371
  specifying input restrictions, 365
  specifying the contents, 372
TextField, 147
timeout job
  canceling, 498
TimeOut predicate
  calling, 391
ToggleButton
  checking the status, 376
  setting the status, 376
ToggleButton gadget, 157
ToggleButton widget, 154
Toolkit
  querying the version, 522
## Index

**Toolkit Intrinsics**
- initializing, 466
- toplevel widget, 21, 512
- toplevel window
  - creating, 399
  - TopLevelShell, 158
  - TransientShell, 159
- translation, 9
- translation table, 9
- translations
  - augment, 394, 484

**UIL, 7, 29**
- assignment of names and values, 205
- assignment of UIL names to callback predicates, 206

**UIL variable**
- fetch value, 198

**UIL widget**
- creating, 201
- naming, 201

**User Interface Language, 7, 29**

**VendorShell, 160**

**widget**
- accept keyboard focus, 396
- ApplicationShell, 37
- ArrowButton, 38
- BulletinBoard, 41
- CascadeButton, 45
- check ability to activate, 473
- check realization, 472
- checking class membership, 474
- checking management, 471
- Command, 48
- Composite, 50
- Constraint, 51
- Core, 52
- creating, 403
- creating from UIL module, 201
- deleting, 519
- destroying, 404
- DialogShell, 55
- Display, 56
- DragContext, 58
- DragIcon, 63

**Draw, 65**
**DrawingArea, 70**
**DrawnButton, 72**
**DropSite, 74**
**FileSelectionBox, 77**
**Form, 80**
**Frame, 85**
**Gadget, 87**
**Label, 90**
**List, 94**
**MainWindow, 99**
**Manager, 101**
**map state, 457**
**mapping, 481**
**MenuShell, 104**
**MessageBox, 105**
**outputting class of, 398**
**OverrideShell, 107**
**PanedWindow, 108**
**place at bottom of widget stack, 478**
**placing at top of window stack, 491**
**Primitive, 111**
**PushButton, 114**
**querying names, 523**
**realizing, 493**
**retrieve toolkit name, 482**
**Screen, 129**
**set cursor, 510**
**set location, 507**
**setting so it can be activated, 506**
**switching off management, 518**

**widget class, 8**
**widget class, opaque, 8**
**widget hierarchy, 7**
**widget instance, 8**
**widget name**
- converting, 483
- querying, 524

**widget tree, creating a, 21**
**widget tree, realizing a, 21**
**widget, creating a, 21**
**widget, managing a, 21**
**widget, RowColumn, 117**
**widget, Scale, 126**
**widget, ScrollBar, 132**
Index

widget, ScrolledWindow, 136
widget, SelectionBox, 139
widget, Separator, 143
widget, Shell, 145
widget, states, 21
widget, Text, 147
widget, TextField, 147
widget, ToggleButton, 154
widget, ToplevelShell, 158
widget, TransientShell, 159
widget, VendorShell, 160
widget, WMShell, 164
widgets, 7
    creating, 244
    releasing, 492
Widgets and gadgets, 35
window, 19
window area
    clearing, 410
    copying, 411
window contents
    repainting, 170
window for a widget
    deleting, 520
window id
    querying, 524
Window Manager, 6
window name
    converting, 483
window stack
    changing the order, 500
Window System, 6
WM protocol
    adding a callback, 213
WM protocol, deleting callbacks, 321
WMShell, 164
work job
    canceling, 499
work predicate
    calling, 393

X, 6
X environment, initializing, 20
X server
    registering an atom, 287
X Toolkit Intrinsics, 6

Xlib, 6
XmFontList, 17
XmString, 16
XmStringTable, 18
Xt Intrinsics, 6
XtAccelerator, 18
XtTranslation, 18