Motif 2.1—Programmer's Reference

Desktop Product Documentation

The Open Group

Desktop Product Documentation:

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

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

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

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

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

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

or by Electronic Mail to:
OGPubs@opengroup.org

OTHER NOTICES

Contents

Preface xvii

The Open Group xvii
The Development of Product Standards xviii
Open Group Publications xix
Versions and Issues of Specifications xxi
Corrigenda xxi
Ordering Information xxi
This Book xxii
Audience xxii
Applicability xxii
Purpose xxii
Organization xxii
Reference Page Format xxiii
Related Documents xxiv
Typographic and Keying Conventions xxv
DocBook SGML Conventions xxvi
Terminology Conventions xxvi
Keyboard Conventions xxvi
Mouse Conventions xxvii
Problem Reporting xxvii
Trademarks xxvii

Chapter 1. Programs 1
mwm 2
uil 37
xmbind 39

Chapter 2. Xt Widget Classes 41
ApplicationShell 42
Composite 47
Constraint 51
Core 54
Object 59
OverrideShell 61
RectObj 64
Shell 67
TopLevelShell 71
TransientShell 76
VendorShell 81
WMShell 94

Chapter 3. Xm Widget Classes 103
XmArrowButton 104
XmArrowButtonGadget 111
XmBulletinBoard 117
XmCascadeButton 128
XmCascadeButtonGadget 139
XmComboBox 147
XmCommand 161
XmContainer 171
XmDialogShell 213
XmDisplay 219
XmDragContext 230
XmDragIcon 252
XmDrawingArea 257
XmDrawnButton 268
XmDropTransfer 277
XmFileSelectionBox 281
XmForm 303
XmFrame 319
XmGadget 325
XmIconGadget 333
XmLabel 340
XmLabelGadget 356
XmList 371
XmMainWindow 406
XmManager 413
XmMenuShell 428
XmMessageBox 434
XmNotebook 444

XmPanedWindow 460
XmPrimitive 470
XmPushButton. 486
XmPushButtonGadget 498
XmRendition 508
XmRowColumn 512
XmScale 540
XmScreen 558
XmScrollBar 567
XmScrolledWindow 583
XmSelectionBox 595
XmSeparator 608
XmSeparatorGadget 613
XmText 618
XmTextField 673
XmToggleButton 709
XmToggleButtonGadget 725

Chapter 4. Translations. 739
VirtualBindings 740

Chapter 5. Xm Data Types 747
XmDirection 748
XmFontList 751
XmParseMapping 754
XmParseTable 759
XmRenderTable 760
XmString 763
XmStringDirection. 766
XmStringTable. 767
XmTab 768
XmTabList. 769
XmTextPosition 771

Chapter 6. Xm Functions 773
XmActivateProtocol 774
XmActivateWMProtocol 776
XmAddProtocolCallback 777
XmAddProtocols 779
XmAddTabGroup 780
XmAddToPostFromList 781
XmAddWMProtocolCallback 783
XmAddWMProtocols 784

XmCascadeButtonGadgetHighlight 785
XmCascadeButtonHighlight 786
XmChangeColor 787
XmClipboardCancelCopy 788
XmClipboardCopy 790
XmClipboardCopyByName 793
XmClipboardEndCopy 795
XmClipboardEndRetrieve 797
XmClipboardInquireCount 799
XmClipboardInquireFormat 801
XmClipboardInquireLength 803
XmClipboardInquirePendingItems 805
XmClipboardLock 807
XmClipboardRegisterFormat 809
XmClipboardRetrieve 811
XmClipboardStartCopy 814
XmClipboardStartRetrieve 817
XmClipboardUndoCopy 819
XmClipboardUnlock 821
XmClipboardWithdrawFormat 823
XmComboBoxAddItem 825
XmComboBoxDeletePos 827
XmComboBoxSelectItem 828
XmComboBoxSetItem 829
XmComboBoxUpdate 830
XmCommandAppendValue 831
XmCommandError 832
XmCommandGetChild 833
XmCommandSetValue 835
XmContainerCopy 836
XmContainerCopyLink 837
XmContainerCut 839
XmContainerGetItemChildren 841
XmContainerPaste 843
XmContainerPasteLink 844
XmContainerRelayout 845
XmContainerReorder 846
XmConvertStringToUnits 847
XmConvertUnits 850
XmCreateArrowButton 853
XmCreateArrowButtonGadget 854
XmCreateBulletinBoard 855
XmCreateBulletinBoardDialog 856
XmCreateCascadeButton 858

XmCreateCascadeButtonGadget 860
XmCreateComboBox 862
XmCreateCommand 863
XmCreateContainer 864
XmCreateDialogShell 865
XmCreateDragIcon 866
XmCreateDrawingArea 868
XmCreateDrawnButton 869
XmCreateDropDownComboBox 870
XmCreateDropDownList 872
XmCreateErrorDialog 873
XmCreateFileSelectionBox 875
XmCreateFileSelectionDialog 877
XmCreateForm 879
XmCreateFormDialog 880
XmCreateFrame 882
XmCreateIconGadget 883
XmCreateInformationDialog 884
XmCreateLabel 886
XmCreateLabelGadget 887
XmCreateList 888
XmCreateMainWindow 889
XmCreateMenuBar 890
XmCreateMenuShell 892
XmCreateMessageBox 893
XmCreateMessageDialog 895
XmCreateNotebook 897
XmCreateOptionMenu 898
XmCreatePanedWindow 901
XmCreatePopupMenu 902
XmCreatePromptDialog 904
XmCreatePulldownMenu 906
XmCreatePushButton 909
XmCreatePushButtonGadget 910
XmCreateQuestionDialog 911
XmCreateRadioBox 913
XmCreateRowColumn 915
XmCreateScale 917
XmCreateScrollBar 918
XmCreateScrolledList 919
XmCreateScrolledText 921
XmCreateScrolledWindow 923
XmCreateSelectionBox. 924
XmCreateSelectionDialog 926

XmCreateSeparator 928
XmCreateSeparatorGadget 929
XmCreateSimpleCheckBox 930
XmCreateSimpleMenuBar 932
XmCreateSimpleOptionMenu 934
XmCreateSimplePopupMenu 936
XmCreateSimplePulldownMenu 938
XmCreateSimpleRadioBox 940
XmCreateSimpleSpinBox 942
XmCreateSpinBox 943
XmCreateTemplateDialog 945
XmCreateText 947
XmCreateTextField 948
XmCreateToggleButton 949
XmCreateToggleButtonGadget 950
XmCreateWarningDialog 951
XmCreateWorkArea 953
XmCreateWorkingDialog 955
XmCvtByteStreamToXmString 957
XmCvtCTToXmString 958
XmCvtStringToUnitType 959
XmCvtTextPropertyToXmStringTable 960
XmCvtXmStringTableToTextProperty 962
XmCvtXmStringToByteStream 964
XmCvtXmStringToCT 966
XmDeactivateProtocol 968
XmDeactivateWMProtocol 970
XmDestroyPixmap 971
XmDirectionMatch 972
XmDirectionMatchPartial 974
XmDirectionToStringDirection 975
XmDragCancel 977
XmDragStart 978
XmDropSite 980
XmDropSiteConfigureStackingOrder 990
XmDropSiteEndUpdate 992
XmDropSiteQueryStackingOrder 993
XmDropSiteRegister 995
XmDropSiteRegistered 997
XmDropSiteRetrieve 998
XmDropSiteStartUpdate 999
XmDropSiteUnregister 1000
XmDropSiteUpdate 1001
XmDropTransferAdd 1002

XmDropTransferStart 1003
XmFileSelectionBoxGetChild 1005
XmFileSelectionDoSearch 1007
XmFontListAdd 1008
XmFontListAppendEntry 1010
XmFontListCopy 1011
XmFontListCreate 1012
XmFontListEntryCreate 1014
XmFontListEntryFree 1016
XmFontListEntryGetFont 1017
XmFontListEntryGetTag 1018
XmFontListEntryLoad 1019
XmFontListFree 1021
XmFontListFreeFontContext 1022
XmFontListGetNextFont 1023
XmFontListInitFontContext 1025
XmFontListNextEntry 1026
XmFontListRemoveEntry 1027
XmGetAtomName 1029
XmGetColorCalculation 1030
XmGetColors 1031
XmGetDestination 1033
XmGetDragContext 1034
XmGetFocusWidget 1036
XmGetMenuCursor 1037
XmGetPixmap 1038
XmGetPixmapByDepth 1042
XmGetPostedFromWidget 1048
XmGetSecondaryResourceData. 1049
XmGetTabGroup 1051
XmGetTearOffControl 1052
XmGetVisibility 1054
XmGetXmDisplay 1056
XmGetXmScreen 1057
XmImCloseXIM 1058
XmImFreeXIC. 1059
XmImGetXIC 1060
XmImGetXIM 1062
XmImMbLookupString 1063
XmImMbResetIC 1066
XmImRegister 1067
XmImSetFocusValues 1068
XmImSetValues 1070
XmImSetXIC 1073

XmImUnregister 1074
XmImUnsetFocus 1075
XmImVaSetFocusValues 1076
XmImVaSetValues 1078
XmInstallImage 1079
XmInternAtom. 1081
XmIsMotifWMRunning 1082
XmIsTraversable 1083
XmListAddItem 1085
XmListAddItemUnselected 1086
XmListAddItems 1087
XmListAddItemsUnselected 1088
XmListDeleteAllItems 1089
XmListDeleteItem 1090
XmListDeleteItems 1091
XmListDeleteItemsPos 1092
XmListDeletePos 1093
XmListDeletePositions 1094
XmListDeselectAllItems 1095
XmListDeselectItem 1096
XmListDeselectPos 1097
XmListGetKbdItemPos. 1098
XmListGetMatchPos 1099
XmListGetSelectedPos 1101
XmListItemExists 1103
XmListItemPos 1104
XmListPosSelected 1105
XmListPosToBounds 1106
XmListReplaceItems 1108
XmListReplaceItemsPos 1110
XmListReplaceItemsPosUnselected. 1112
XmListReplaceItemsUnselected 1114
XmListReplacePositions 1116
XmListSelectItem 1118
XmListSelectPos 1119
XmListSetAddMode 1120
XmListSetBottomItem 1121
XmListSetBottomPos 1122
XmListSetHorizPos 1123
XmListSetItem. 1124
XmListSetKbdItemPos 1125
XmListSetPos 1126
XmListUpdateSelectedList 1127
XmListYToPos 1128

XmMainWindowSep1 1129
XmMainWindowSep2 1130
XmMainWindowSep3 1131
XmMainWindowSetAreas 1132
XmMapSegmentEncoding 1134
XmMenuPosition 1135
XmMessageBoxGetChild 1136
XmNotebookGetPageInfo 1138
XmObjectAtPoint 1141
XmOptionButtonGadget 1143
XmOptionLabelGadget 1145
XmParseMappingCreate 1147
XmParseMappingFree 1148
XmParseMappingGetValues 1149
XmParseMappingSetValues 1150
XmParseTableFree 1151
XmGetScaledPixmap 1152
XmPrintPopupPDM 1154
XmPrintSetup 1157
XmPrintShell 1160
XmPrintToFile 1166
XmProcessTraversal 1169
XmRedisplayWidget 1175
XmRegisterSegmentEncoding 1179
XmRemoveFromPostFromList 1181
XmRemoveProtocolCallback 1183
XmRemoveProtocols 1185
XmRemoveTabGroup 1187
XmRemoveWMProtocolCallback 1188
XmRemoveWMProtocols 1189
XmRenderTableAddRenditions 1190
XmRenderTableCopy 1192
XmRenderTableCvtFromProp 1193
XmRenderTableCvtToProp 1194
XmRenderTableFree 1195
XmRenderTableGetRendition 1196
XmRenderTableGetRenditions 1197
XmRenderTableGetTags 1199
XmRenderTableRemoveRenditions 1200
XmRenditionCreate 1202
XmRenditionFree 1204
XmRenditionRetrieve 1205
XmRenditionUpdate 1206
XmRepTypeAddReverse 1207

XmRepTypeGetId 1208
XmRepTypeGetNameList 1209
XmRepTypeGetRecord 1210
XmRepTypeGetRegistered 1212
XmRepTypeInstallTearOffModelConverter 1214
XmRepTypeRegister 1215
XmRepTypeValidValue 1217
XmResolveAllPartOffsets 1218
XmResolvePartOffsets 1222
XmScaleGetValue 1225
XmScaleSetTicks 1226
XmScaleSetValue 1228
XmScrollBarGetValues 1229
XmScrollBarSetValues 1231
XmScrollVisible 1233
XmScrolledWindowSetAreas 1235
XmSelectionBoxGetChild 1237
XmSetColorCalculation 1239
XmSetFontUnit 1241
XmSetFontUnits 1242
XmSetMenuCursor 1244
XmSetProtocolHooks 1245
XmSetWMProtocolHooks 1247
XmSpinBox 1249
XmSimpleSpinBoxAddItem 1269
XmSimpleSpinBoxDeletePos 1270
XmSimpleSpinBoxSetItem 1271
XmSpinBoxValidatePosition 1272
XmSimpleSpinBox 1276
XmStringBaseline 1285
XmStringByteCompare 1286
XmStringByteStreamLength 1288
XmStringCompare 1289
XmStringComponentCreate 1290
XmStringComponentType 1292
XmStringConcat 1295
XmStringConcatAndFree 1296
XmStringCopy 1298
XmStringCreate 1299
XmStringCreateLocalized 1301
XmStringCreateLtoR 1302
XmStringCreateSimple 1304
XmStringDirectionCreate 1306
XmStringDirectionToDirection 1307

XmStringDraw 1308
XmStringDrawImage 1310
XmStringDrawUnderline 1312
XmStringEmpty 1314
XmStringExtent 1315
XmStringFree 1316
XmStringFreeContext 1317
XmStringGenerate 1318
XmStringGetLtoR 1320
XmStringGetNextComponent 1322
XmStringGetNextSegment 1325
XmStringGetNextTriple 1327
XmStringHasSubstring 1329
XmStringHeight 1330
XmStringInitContext 1331
XmStringIsVoid 1332
XmStringLength 1333
XmStringLineCount 1334
XmStringNConcat 1335
XmStringNCopy 1337
XmStringParseText. 1338
XmStringPeekNextComponent 1341
XmStringPeekNextTriple 1342
XmStringPutRendition 1343
XmStringSegmentCreate 1345
XmStringSeparatorCreate 1347
XmStringTableParseStringArray 1348
XmStringTableProposeTablist 1350
XmStringTableToXmString. 1352
XmStringTableUnparse. 1354
XmStringToXmStringTable. 1356
XmStringUnparse 1358
XmStringWidth 1361
XmTabCreate 1362
XmTabFree 1364
XmTabGetValues 1365
XmTabListCopy 1367
XmTabListFree 1369
XmTabListGetTab 1370
XmTabListInsertTabs 1371
XmTabListRemoveTabs 1373
XmTabListReplacePositions 1375
XmTabListTabCount 1377
XmTabSetValue 1378

XmTargetsAreCompatible 1379
XmTextClearSelection 1381
XmTextCopy 1382
XmTextCopyLink 1384
XmTextCut 1386
XmTextDisableRedisplay 1388
XmTextEnableRedisplay 1389
XmTextFieldClearSelection 1390
XmTextFieldCopy 1391
XmTextFieldCopyLink 1392
XmTextFieldCut 1394
XmTextFieldGetBaseline 1396
XmTextFieldGetEditable 1397
XmTextFieldGetInsertionPosition 1398
XmTextFieldGetLastPosition 1399
XmTextFieldGetMaxLength 1400
XmTextFieldGetSelection 1401
XmTextFieldGetSelectionPosition 1402
XmTextFieldGetSelectionWcs 1404
XmTextFieldGetString 1405
XmTextFieldGetStringWcs 1406
XmTextFieldGetSubstring 1407
XmTextFieldGetSubstringWcs 1409
XmTextFieldInsert 1411
XmTextFieldInsertWcs 1413
XmTextFieldPaste 1415
XmTextFieldPasteLink 1417
XmTextFieldPosToXY 1418
XmTextFieldRemove 1420
XmTextFieldReplace 1421
XmTextFieldReplaceWcs 1423
XmTextFieldSetAddMode 1425
XmTextFieldSetEditable 1426
XmTextFieldSetHighlight 1427
XmTextFieldSetInsertionPosition 1429
XmTextFieldSetMaxLength 1430
XmTextFieldSetSelection 1431
XmTextFieldSetString 1432
XmTextFieldSetStringWcs 1433
XmTextFieldShowPosition 1434
XmTextFieldXYToPos 1435
XmTextFindString 1436
XmTextFindStringWcs 1438
XmTextGetBaseline 1440

XmTextGetCenterline 1441
XmTextGetEditable 1442
XmTextGetInsertionPosition 1443
XmTextGetLastPosition 1444
XmTextGetMaxLength 1445
XmTextGetSelection 1446
XmTextGetSelectionPosition 1447
XmTextGetSelectionWcs 1449
XmTextGetSource 1450
XmTextGetString 1451
XmTextGetStringWcs 1452
XmTextGetSubstring 1453
XmTextGetSubstringWcs 1455
XmTextGetTopCharacter 1457
XmTextInsert 1458
XmTextInsertWcs 1460
XmTextPaste 1462
XmTextPasteLink 1464
XmTextPosToXY 1465
XmTextRemove 1467
XmTextReplace 1468
XmTextReplaceWcs 1470
XmTextScroll 1472
XmTextSetAddMode 1473
XmTextSetEditable 1474
XmTextSetHighlight 1475
XmTextSetInsertionPosition 1477
XmTextSetMaxLength 1478
XmTextSetSelection 1479
XmTextSetSource 1480
XmTextSetString 1482
XmTextSetStringWcs 1483
XmTextSetTopCharacter 1484
XmTextShowPosition 1485
XmTextXYToPos 1486
XmToggleButtonGadgetGetState 1488
XmToggleButtonGadgetSetState 1489
XmToggleButtonGetState 1491
XmToggleButtonSetState 1492
XmToggleButtonSetValue 1493
XmTrackingEvent 1494
XmTrackingLocate 1496
XmTransferDone 1498
XmTransferSendRequest 1501

XmTransferSetParameters 1502
XmTransferStartRequest 1504
XmTransferValue 1505
XmTranslateKey 1508
XmUninstallImage 1510
XmUpdateDisplay 1511
XmVaCreateSimpleCheckBox 1512
XmVaCreateSimpleMenuBar 1515
XmVaCreateSimpleOptionMenu 1517
XmVaCreateSimplePopupMenu 1521
XmVaCreateSimplePulldownMenu 1525
XmVaCreateSimpleRadioBox 1529
XmWidgetGetBaselines 1532
XmWidgetGetDisplayRect 1534

Chapter 7. Mrm Functions 1537
MrmCloseHierarchy 1538
MrmFetchBitmapLiteral 1539
MrmFetchColorLiteral 1541
MrmFetchIconLiteral 1543
MrmFetchLiteral 1545
MrmFetchSetValues 1547
MrmFetchWidget 1549
MrmFetchWidgetOverride 1551
MrmInitialize 1553
MrmOpenHierarchy 1554
MrmOpenHierarchyFromBuffer 1558
MrmOpenHierarchyPerDisplay 1560
MrmRegisterClass 1565
MrmRegisterNames 1567
MrmRegisterNamesInHierarchy 1569

Chapter 8. Uil Functions 1571
Uil 1572
UilDumpSymbolTable 1578

Chapter 9. File Formats 1581
mwmrc 1582
Traits 1597
UIL 1606
WML 1642

Appendix A. Constraint Arguments and Automatically Created Children 1653

Appendix B. UIL Built-In Tables 1659

Appendix C. UIL Arguments 1755

Appendix D. UIL Diagnostic Messages 1773

Preface

The Open Group

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

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

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

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

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

The Development of Product Standards

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

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

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

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

Open Group Publications

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

There are several types of specification:

CAE Specifications

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

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

Preliminary Specifications

Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

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

Consortium and Technology Specifications

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

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

In addition, The Open Group publishes:

Product Documentation

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

Guides

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

Technical Studies

Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

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

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

Corrigenda

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

Ordering Information

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

This Book

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

Audience

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

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

Applicability

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

Purpose

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

Organization

This document is organized into nine chapter and four appendixes:

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

Reference Page Format

The reference pages in this volume use the following format:

Purpose This section gives a short description of the interface.

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

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

Name	Contains the name of the resource. Each new resource is described following the new resources table.
Class	Contains the class of the resource.
Type	Contains the type of the resource.
Default	Contains the default value of the resource.

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

Examples This section gives practical examples for using the interface.

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

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

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

Related Documents

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Xlib—C Language X Interface

X Toolkit Intrinsics—C Language Interface

Typographic and Keying Conventions

This book uses the following conventions.

DocBook SGML Conventions

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

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

Terminology Conventions

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

Keyboard Conventions

Because not all keyboards are the same, it is difficult to specify keys that are correct for every manufacturer’s keyboard. To solve this problem, this guide describes keys that use a *virtual key* mechanism. The term *virtual* implies that the keys as described do not necessarily correspond to a fixed set of actual keys. Instead, virtual keys are linked to actual keys by means of *virtual bindings*. A given virtual key may be bound to different physical keys for different keyboards.

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

Mouse Conventions

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

Problem Reporting

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

Trademarks

Motif® OSF/1®, and UNIX® are registered trademarks and the IT DialTone™, The Open Group™, and the ''X Device''™ are trademarks of The Open Group.

AIX is a trademark of International Business Machines Corp.

HP/UX is a trademark of Hewlett Packard Company.

Solaris is a trademark of Sun Microsystems, Inc.

Chapter 4
Translations

VirtualBindings

Purpose

Bindings for virtual mouse and key events

Description

The Motif reference pages describe key translations in terms of *virtual bindings*, based on those described in the *CDE 2.1/Motif 2.1—Style Guide and Glossary*.

Bindings for osf Keysyms

Keysym strings that begin with osf are not part of the X server’s keyboard mapping. Instead, these keysyms are produced on the client side at run time. They are interpreted by the routine **XmTranslateKey**, and are used by the translation manager when the server delivers an actual key event. For each application, a mapping is maintained between osf keysyms and keysyms that correspond to actual keys. This mapping is based on information obtained at application startup from one of the following sources, listed in order of precedence:

- The **XmNdefaultVirtualBindings** resource from Display.
- A property on the root window, which can be set by **mwm** on startup, or by the **xmbind** client, or on prior startup of a Motif application.
- The file **.motifbind** in the user’s home directory.
- A set of bindings based on the vendor string and optionally the vendor release of the X server. Motif searches for these bindings in the following steps:
  1. If the file **xmbind.alias** exists in the user’s home directory, Motif searches this file for a pathname associated with the vendor string or with the vendor string and vendor release. If it finds such a pathname and if that file exists, Motif loads the bindings contained in that file.
  2. If it has found no bindings, Motif next looks for the file **xmbind.alias** in the directory specified by the environment variable **XMBINDDIR**, if **XMBINDDIR** is set, or in the directory **/usr/lib/Xm/bindings** if **XMBINDDIR** is not set. If this file exists Motif searches it for a pathname associated with the vendor string or with the vendor string and vendor

release. If it finds such a pathname and if that file exists, Motif loads the bindings contained in that file.

3. If it still has found no bindings, Motif loads a set of hard-coded fallback bindings.

The **xmbind.alias** file contains zero or more lines of the following form:

```
"vendor_string[ vendor_release]"    bindings_file
```

where *vendor_string* is the X server vendor name as returned by the X client **xdpyinfo** or the Xlib function **XServerVendor**, and must appear in double quotes. If *vendor_release* is included, it is the X server vendor release number as returned by the X client **xdpyinfo** or the Xlib function **XVendorRelease**, and must also be contained within the double quotes separated by one space from *vendor_string*. The *vendor_release* argument is provided to allow support of changes in keyboard hardware from a vendor, assuming that the vendor increments the release number to flag such changes. Alternatively, the vendor may simply use a unique vendor string for each different keyboard.

The *bindings_file* argument is the pathname of the file containing the bindings themselves. It can be a relative or absolute pathname. If it it is a relative pathname, it is relative to the location of the **xmbind.alias** file.

Comment lines in the **xmbind.alias** file begin with ! (exclamation point).

The bindings found in either the **.motifbind** file or the vendor mapping are placed in a property on the root window. This property is used to determine the bindings for subsequent Motif applications.

On startup **mwm** attempts to load the file **.motifbind** in the user's home directory. If this is unsuccessful, it loads the vendor bindings as described previously. It places the bindings it loads in a property on the root window for use by subsequent Motif applications.

The **xmbind** function loads bindings from a file if that file is specified on the command line. If no file is specified on the command line, it attempts to load the file **.motifbind** in the user's home directory. If this fails, it loads the vendor bindings as described previously. It places the bindings it loads in a property on the root window for use by subsequent Motif applications.

The format of the specification for mapping osf keysyms to actual keysyms is similar to that of a specification for an event translation. (See below) The syntax is specified (and below) here in EBNF notation using the following conventions:

```
[a]    Means either nothing or a
{a}    Means zero or more occurrences of a
(a|b)  Means either a or b.
```

Terminals are enclosed in double quotation marks.

The syntax of an osf keysym binding specification is as follows:

```
binding_spec      =      {line "\n"} [line]
line              =     virtual_keysym ":" list_of_key_event
list_of_key_event =     key_event { "," key_event}
key_event         =     {modifier_name} "<Key>" actual_keysym
virtual_keysym    =     keysym
actual_keysym     =     keysym
keysym            =     A valid X11 keysym name that is
                        mapped by XStringToKeysym
```

As with event translations, more specific event descriptions must precede less specific descriptions. For example, an event description for a key with a modifier must precede a description for the same key without the same modifier.

Following is an example of a specification for the **defaultVirtualBindings** resource in a resource file:

```
*defaultVirtualBindings: \
        osfBackSpace   :       <Key>BackSpace       \n\
        osfInsert      :       <Key>InsertChar      \n\
        osfDelete      :       <Key>DeleteChar      \n\
...
        osfLeft        :       <Key>left, Ctrl<Key>H
```

The format of a **.motifbind** file or of a file containing vendor bindings is the same, except that the binding specification for each keysym is placed on a separate line. The previous example specification appears as follows in a **.motifbind** or vendor bindings file:

```
osfBackSpace           :       <Key>BackSpace
osfInsert              :       <Key>InsertChar
osfDelete              :       <Key>DeleteChar
...
osfLeft                :       <Key>left, Ctrl<Key>H
```

The following table lists the fixed fallback default bindings for **osf** keysyms.

Fallback Default Bindings for osf Keysyms	
osf Keysym	**Fallback Default Binding**
osfActivate:	**<Key>KP_Enter <Key>Execute**
osfAddMode:	**Shift<Key>F8**
osfBackSpace:	**<Key>BackSpace**
osfBeginLine:	**<Key>Home <Key>Begin**
osfCancel:	**<Key>Escape <Key>Cancel**
osfClear:	**<Key>Clear**
osfCopy:	*unbound*
osfCut:	*unbound*
osfDelete:	**<Key>Delete**
osfDeselectAll:	*unbound*
osfDown:	**<Key>Down**
osfEndLine:	**<Key>End**
osfHelp:	**<Key>F1 <Key>Help**
osfInsert:	**<Key>Insert**
osfLeft:	**<Key>Left**
osfLeftLine:	*unbound*
osfMenu:	**Shift<Key>F10 <Key>Menu**
osfMenuBar:	**<Key>F10 Shift<Key>Menu**
osfNextMinor:	*unbound*
osfPageDown:	**<Key>Next**
osfPageLeft:	*unbound*
osfPageRight:	*unbound*
osfPageUp:	**<Key>Prior**
osfPaste:	*unbound*

osfPrimaryPaste:	*unbound*
osfPriorMinor:	*unbound*
osfReselect:	*unbound*
osfRestore:	*unbound*
osfRight:	**<Key>Right**
osfRightLine:	*unbound*
osfSelect:	**<Key>Select**
osfSelectAll:	*unbound*
osfSwitchDirection:	**Alt<Key>Return Alt<Key>KP_Enter**
osfUndo:	**<Key>Undo**
osfUp:	**<Key>Up**

Changes in the Handling of Shifted Keys

In conjunction with MIT X11R5 Patch 24, this version of Motif introduces a change in the way that keys involving the <Shift> modifier are processed. This change allows the numeric keypad to be used to generate numbers using the standard X mechanisms. Since the default behavior is now to honor the xmodmap keymap bindings, translations and virtual key bindings that use <Shift> may behave differently. A common symptom is that unshifted keypad and function keys (with or without other modifiers) produce the expected results, but shifted ones do not.

To obtain the old behavior you can remove the shifted interpretation from problematic keys using the **xmodmap** utility. Each entry in a **xmodmap** keymap table contains up to four keysym bindings. The second and fourth keysyms are for shifted keys. If an expression contains only two keysyms, simply remove the second keysym. If an entry contains three or more keysyms, replace the second keysym with **NoSymbol** and remove the fourth keysym.

Action Translations

The translation table syntax used by Motif is completely specified in the X11R5 Toolkit Intrinsics Documentation. For the complete syntax description, and for general instructions about writing or modifying a translation table, please refer to this document. A brief summary of the translation table format, however, is included below.

The syntax is defined as in the binding syntax specification above. Informal descriptions are contained in angle brackets (<>).

```
TranslationTable =    [ directive ] { production }
directive        =    ( "#replace" | "#override" | "#augment") "\n"
production       =    lhs ":" rhs "\n"
lhs              =    ( event | keyseq) {"," ( event | keyseq) }
keyseq           =    """ keychar { keychar } """
keychar          =    ( "^" | "$" | "\\") <ISO Latin 1 character>
event            =    [ modifier_list ] "<" event_type ">" [ count ] {detail}
modifier_list    =    ( ["!"][":"] { modifier } | "None")
modifier         =    [ "~" ] ( "@" <keysym> | <name from table below>)
count            =    "(" <positive integer> [ "+" ] ")"
rhs              =    { action_name "(" [params] ")" }
params           =    string { "," string }
```

The *string* field need not be quoted unless it includes a space or tab character, or any comma, newline, or parenthesis. The entire list of string values making up the *params* field will ba passed to the named action routine.

The *details* field may be used to specify a keysym that will identify a particular key event. For example, Key is the name of a type of event, but it must be modified by the *details* field to name a specific event, such as Key**A**.

Modifier Names The modifier list, which may be empty, consists of a list of modifier keys that must be pressed with the key sequence. The modifier keys may abbreviated with single letters, as in the following list of the familiar modifiers:

s	Shift
c or **^**	Ctrl (Control)
m or **$**	Meta
a	Alt

Other modifiers are available, such as "Mod5" and "Button2." These have no abbreviation (although the "Button" modifiers may be abbreviated in combination with events, as outlined below). If a modifier list has no entries, and is not "None", it means the position of the modifier keys is irrelevant. If modifiers are listed, the designated keys must be in the specified position, but the unlisted modifier keys are irrelevant. If the list begins with an exclamation point (!), however, the unlisted modifiers may not be asserted. In addition, if a modifier name is preceded by a tilde (~), the corresponding key must *not* be pressed.

If a modifier list begins with a colon (:), X tries to use the standard modifiers (Shift and Lock), if present, to map the key event code into a recognized keysym.

Event Types These are a few of the recognized event types.

Key or KeyDown	A keyboard key was pressed.
KeyUp	A keyboard key was released.
BtnDown	A mouse button was pressed.
BtnUp	A mouse button was released.
Motion	The mouse pointer moved.
Enter	The pointer entered the widget's window.
Leave	The pointer left the widget's window.
FocusIn	The widget has received focus.
FocusOut	The widget has lost focus.

There are some event abbreviations available. For example, Btn1Motion is actually a "Motion" event, modified with the "Button1" modifier (**Button1<Motion>**). Similarly, Btn3Up is actually a "BtnUp" event with the "Button3" modifier. These abbreviations are used extensively in the Motif translation tables.

Related Information

xmbind(1)

Chapter 5

Xm Data Types

XmDirection

Purpose

Data type for the direction of widget components

Synopsis

#include <Xm/Xm.h>

Description

XmDirection is the data type specifying the direction in which the system displays subwidgets, children of widgets, or other visual components that are to be laid out. This data type also affects traversal order within tab groups.

XmDirection is implemented as an unsigned char bit mask. The horizontal and vertical directions can be specified independent of each other. **XmDirection** also specifies the precedence of the horizontal and vertical directions relative to each other. For example, a value of **XmRIGHT_TO_LEFT_TOP_TO_BOTTOM** lays out a component horizontally from right to left first, then vertically top to bottom.

XmDirection provides the following masks, each of which corresponds to a particular bit in **XmDirection**:

- **XmRIGHT_TO_LEFT_MASK**
- **XmLEFT_TO_RIGHT_MASK**
- **XmTOP_TO_BOTTOM_MASK**
- **XmBOTTOM_TO_TOP_MASK**
- **XmPRECEDENCE_HORIZ_MASK**
- **XmPRECEDENCE_VERT_MASK**

In addition to the preceding single bit masks, **XmDirection** also provides the following multiple bit masks. These multiple bit masks are particularly useful as arguments to **XmDirectionMatchPartial**:

- **XmHORIZONTAL_MASK**
- **XmPRECEDENCE_MASK**

- **XmVERTICAL_MASK**

Motif also provides the following enumerated constants for specifying various combinations of directions:

XmRIGHT_TO_LEFT_TOP_TO_BOTTOM
Specifies that the components are laid out from right to left first, then top to bottom.

XmLEFT_TO_RIGHT_TOP_TO_BOTTOM
Specifies that the components are laid out from left to right first, then top to bottom.

XmRIGHT_TO_LEFT_BOTTOM_TO_TOP
Specifies that the components are laid out from right to left first, then bottom to top.

XmLEFT_TO_RIGHT_BOTTOM_TO_TOP
Specifies that the components are laid out from left to right first, then bottom to top.

XmTOP_TO_BOTTOM_RIGHT_TO_LEFT
Specifies that the components are laid out from top to bottom first, then right to left.

XmTOP_TO_BOTTOM_LEFT_TO_RIGHT
Specifies that the components are laid out from top to bottom first, then left to right.

XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
Specifies that the components are laid out from bottom to top first, then right to left.

XmBOTTOM_TO_TOP_LEFT_TO_RIGHT
Specifies that the components are laid out from bottom to top first, then left to right.

XmTOP_TO_BOTTOM
Specifies that the components are laid out from top to bottom. If horizontal direction is important, do not use this constant.

XmBOTTOM_TO_TOP
Specifies that the components are laid out from bottom to top. If horizontal direction is important, do not use this constant.

XmDEFAULT_DIRECTION
Specifies that the components are laid out according to the default direction. (This constant is primarily for widget writers.)

XmLEFT_TO_RIGHT
Specifies that the components are laid out from left to right. If vertical direction is important, do not use this constant.

XmRIGHT_TO_LEFT
Specifies that the components are laid out from right to left. If vertical direction is important, do not use this constant.

Related Information

XmDirectionMatch(3), **XmDirectionMatchPartial**(3), **XmDirectionToStringDirection**(3), **XmString**(3), **XmStringDirection**(3), and **XmStringDirectionToDirection**(3).

XmFontList

Purpose

Data type for a font list

Synopsis

#include <Xm/Xm.h>

Description

XmFontList is the data type for a font list. A font list consists of font list entries. Each entry contains a font or a font set (a group of fonts) and is identified with a tag, which is optional. If this tag is NULL, the tag is set to **XmFONTLIST_DEFAULT_TAG**.

The value of **XmFONTLIST_DEFAULT_TAG** is **XmFONTLIST_DEFAULT_TAG_STRING**.

When a compound string is displayed, the font list element tag of the compound string segment is matched with a font list entry tag in the font list and the matching font list entry is used to display the compound string. A font list entry is chosen as follows:

- The first font list entry whose tag matches the tag of the compound string segment is used.
- If no match has been found and if the tag of the compound string segment is **XmFONTLIST_DEFAULT_TAG**, the first font list entry whose tag matches the tag that would result from creating an entry with **XmSTRING_DEFAULT_CHARSET** is used. For example, if creating an entry with **XmSTRING_DEFAULT_CHARSET** would result in the tag **ISO8859-1**, the compound string segment tag **XmFONTLIST_DEFAULT_TAG** matches the font list entry tag **ISO8859-1**.
- If no match has been found and if the tag of the compound string segment matches the tag that would result from creating a segment with **XmSTRING_DEFAULT_CHARSET**, the first font list entry whose tag is **XmFONTLIST_DEFAULT_TAG** is used.
- If no match has been found, the first entry in the font list is used.

The font list interface consists of the routines listed in **Related Information**.

Font lists are specified in resource files with the following syntax:

resource_spec: *font_entry* [, *font_entry*]+

The resource value string consists of one or more font list entries separated by commas. Each *font_entry* identifies a font or font set and an optional font list entry tag. A tag specified for a single font follows the font name and is separated by = (equals sign); otherwise, in a font set the tag is separated by a colon. The colon is required whether a tag is specified or not. A font entry uses the following syntax to specify a single font:

font_name ['=' *tag*]

For example, the following entry specifies a 10 point Times Italic font without a font list entry tag;

```
*fontList:  -Adobe-Times-Medium-I-Normal--10*
```

A font entry containing a font set is similar, except a semicolon separates multiple font names and the specification ends with a colon followed by an optional tag:

font_name [';' *font_name*]+ ':' [*tag*]

A *font_name* is an X Logical Font Description (XLFD) string and *tag* is any set of characters from ISO646IRV except space, comma, colon, equal sign and semicolon. Following is an example of a font set entry. It consists of three fonts (except for charsets), and an explicit font list entry tag.

```
*fontList: -Adobe-Courier-Bold-R-Normal--25-180-100-100-M-150;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-240;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-120:MY_TAG
```

Note that the **XmRenderTable** is another data type that can be used for font lists. Refer to the **XmRenderTable**(3) for details.

Related Information

XmFontListAdd(3), **XmFontListAppendEntry**(3), **XmFontListCopy**(3), **XmFontListCreate**(3), **XmFontListEntryCreate**(3), **XmFontListEntryFree**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryGetTag**(3), **XmFontListEntryLoad**(3), **XmFontListFree**(3), **XmFontListFreeFontContext**(3),

XmFontListGetNextFont(3), **XmFontListInitFontContext**(3), **XmFontListNextEntry**(3), **XmFontListRemoveEntry**(3), **XmRenderTable**(3), and **XmString**(3).

XmParseMapping

Purpose

Data type for a compound string parse mapping

Synopsis

#include <Xm/Xm.h>

Description

XmParseMapping is an opaque data type for a parse mapping used by **XmStringParseText** to create a compound string. A parse mapping contains a pattern to be matched in text being parsed to create a compound string. It also contains a compound string, or a function to be invoked to provide a compound string, to be included in the compound string being created whenever the pattern is matched.

An application uses a resource-style interface to specify components for an **XmParseMapping**. **XmParseMappingCreate** creates a parse mapping, using a resource-style argument list. **XmParseMappingGetValues** and **XmParseMappingSetValues** retrieve and set the components of a parse mapping. **XmParseMappingFree** recovers memory used by a parse mapping. **XmParseTable** is an array of **XmParseMapping** objects.

The **XmNinvokeParseProc** resource is a function of type **XmParseProc**, which is defined as follows:

XmIncludeStatus (*XmParseProc) (*text_in_out, text_end, type, tag, entry, pattern_length, str_include, call_data*)
    XtPointer **text_in_out*;
    XtPointer *text_end*;
    XmTextType *type*;
    XmStringTag *tag*;
    XmParseMapping *entry*;
    int *pattern_length*;
    XmString **str_include*;
    XtPointer *call_data*;

A parse procedure provides an escape mechanism for arbitrarily complex parsing. This procedure is invoked when a pattern in the input text is matched with a pattern in a parse mapping whose **XmNincludeStatus** is **XmINVOKE**.

The input text is a pointer to the first byte of the pattern that was matched to trigger the call to the parse procedure. The parse procedure consumes as many bytes of the input string as it needs and sets the input text pointer to the following byte. It returns a compound string to be included in the compound string being constructed, and it also returns an **XmIncludeStatus** indicating how the returned compound string should be handled. If the parse procedure does not set the input text pointer ahead by at least one byte, the parsing routine continues trying to match the input text with the patterns in the remaining parse mappings in the parse table. Otherwise, the parsing routine begins with the new input text pointer and tries to match the input text with patterns in the parse mappings starting at the beginning of the parse table.

text_in_out	Specifies the text being parsed. The value is a pointer to the first byte of text matching the pattern that triggered the call to the parse procedure. When the parse procedure returns, this argument is set to the position in the text where parsing should resume—that is, to the byte following the last character parsed by the parse procedure.
text_end	Specifies a pointer to the end of the *text_in_out* string. If *text_end* is NULL, the string is scanned until a NULL character is found. Otherwise, the string is scanned up to but not including the character whose address is *text_end*.
type	Specifies the type of text and the tag type. If a locale tag should be created, *type* has a value of either **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**. If a charset should be created, *type* has a value of **XmCHARSET_TEXT**.
tag	Specifies the tag to be used in creating the result. The type of string tag created (charset or locale) depends on the text type and the passed in *tag* value. If the *tag* value is NULL and if *type* indicates that a charset string tag should be created, the string tag has the value that is the result of mapping **XmSTRING_DEFAULT_CHARSET**. If *type* indicates a locale string tag, the string tag has the value **_MOTIF_DEFAULT_LOCALE**.
entry	Specifies the parse mapping that triggered the call to the parse procedure.

pattern_length
Specifies the number of bytes in the input text, following *text_in_out*, that constitute the matched pattern.

str_include Specifies a pointer to a compound string. The parse procedure creates a compound string to be included in the compound string being constructed. The parse procedure then returns the compound string in this argument.

call_data Specifies data passed by the application to the parsing routine.

The parse procedure returns an **XmIncludeStatus** indicating how *str_include* is to be included in the compound string being constructed. Following are the possible values:

XmINSERT Concatenate the result to the compound string being constructed and continue parsing.

XmTERMINATE
Concatenate the result to the compound string being constructed and terminate parsing.

New Resources

The following table defines a set of resources used by the programmer to specify data. The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XmParseMappingSetValues** (S), retrieved by using **XmParseMappingGetValues** (G), or is not applicable (N/A).

XmParseMapping Resource Set

Name	Class	Type	Default	Access
XmNclientData		XtPointer	NULL	CSG
XmNincludeStatus		XmIncludeStatus	XmINSERT	CSG
XmNinvokeParseProc		XmParseProc	NULL	CSG
XmNpattern		XtPointer	NULL	CSG
XmNpatternType		XmTextType	XmCHARSET_TEXT	CSG
XmNsubstitute		XmString	NULL	CSG

XmNclientData
Specifies data to be used by the parse procedure.

XmNincludeStatus

Specifies how the result of the mapping is to be included in the compound string being constructed. Unless the value is **XmINVOKE**, the result of the mapping is the value of **XmNsubstitute**. Following are the possible values for **XmNincludeStatus**:

XmINSERT Concatenate the result to the compound string being constructed and continue parsing.

XmINVOKE

Invoke the **XmNinvokeParseProc** on the text being parsed and use the returned compound string instead of **XmNsubstitute** as the result to be inserted into the compound string being constructed. The include status returned by the parse procedure (**XmINSERT** or **XmTERMINATE**) determines how the returned compound string is included.

XmTERMINATE

Concatenate the result to the compound string being constructed and terminate parsing.

XmNinvokeParseProc

Specifies the parse procedure to be invoked when **XmNincludeStatus** is **XmINVOKE**.

XmNpattern

Specifies a pattern to be matched in the text being parsed. This is a maximum of one character.

XmNpatternType

Specifies the type of the pattern that is the value of **XmNpattern**. Following are the possible values:

- **XmCHARSET_TEXT**
- **XmMULTIBYTE_TEXT**
- **XmWIDECHAR_TEXT**

XmNsubstitute

Specifies the compound string to be included in the compound string being constructed when **XmNincludeStatus** is **XmINSERT** or **XmTERMINATE**.

Related Information

XmParseMappingCreate(3), **XmParseMappingFree**(3), **XmParseMappingGetValues**(3), **XmParseMappingSetValues**(3), **XmParseTable**(3), and **XmString**(3).

XmParseTable

Purpose

Data type for a compound string parse table

Synopsis

#include <Xm/Xm.h>

Description

XmParseTable is the data type for an array of parse mappings (objects of type **XmParseMapping**).

A parse table is used by some routines that parse and unparse compound strings. The table is an ordered list of parse mappings. A parsing routine that uses a parse table scans the input text and searches the parse mappings, in order, for one containing a pattern that matches the input text. The matching parse mapping supplies a compound string to be included in the compound string under construction.

An unparsing routine that uses a parse table searches the parse mappings, in order, for one containing a compound string that matches the input compound string. The unparsing routine can then include the parse mapping's text pattern in the output text under construction.

Related Information

XmParseMapping(3), **XmParseTableFree**(3), and **XmString**(3).

XmRenderTable

Purpose

Data type for a render table

Synopsis

#include <Xm/Xm.h>
XmRenderTable

Description

XmRenderTable is the data type for a render table, which contains a table of renditions (**XmRendition**s). Each rendition consists of a set of attributes for rendering text, including a font or fontset, colors, tabs, and lines. Each rendition is identified with a tag.

When a compound string is displayed, for each segment in the string, the rendition used to render that string is formed as follows. If the segment has at least one rendition begin tag or if the list of tags formed by accumulating from previous segments the rendition begin tags and removing the rendition end tags is not empty, these tags are matched with renditions in the render table. The effective rendition used to render the segment is formed by successively merging each rendition found into the current rendition with non-**XmAS_IS** (**XmUNSPECIFIED_PIXEL** for color resources) values for resources in the rendition to be merged, thus replacing the corresponding values of the resources in the current rendition. Finally, if the resulting rendition still has resources with unspecified values and the segment has a locale or charset tag (these are optional and mutually exclusive) this tag is matched with a rendition in the render table, and the missing rendition values are filled in from that entry.

If no matching rendition is found for a particular tag, then the **XmNnoRenditionCallback** of the **XmDisplay** object is called and the render table is searched again for that tag.

If the resulting rendition does not specify a font or fontset, then for segments whose text type is **XmCHARSET_TEXT**, the render table will be searched for a rendition tagged with **XmFONTLIST_DEFAULT_TAG**, and if a matching rendition is found, it will be merged into the current rendition. If the resulting rendition contains no font or fontset, the **XmNnoFontCallback** will be called with the default rendition and "" as the font name. If no rendition matches or no font was found after the callback, then the first rendition in the render table will be merged into the current rendition. If this rendition still has no font, then the segment will not be rendered and a warning will be emitted.

For segments whose text type is **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**, the render table will be searched for a rendition tagged with **_MOTIF_DEFAULT_LOCALE**, and, if a matching rendition is found, it will be merged into the current rendition. If the resulting rendition contains no font, the **XmNnoFontCallback** will be called with the default rendition and "" as the font name. An application can have this callback attempt to remedy this problem by calling **XmRenditionUpdate** on the input rendition to provide a font for the widget to use. This may be done by either providing an alternative font name to be loaded using the **XmNfontName** and **XmNfontType** resources or with an already loaded font using the **XmNfont** resource. If no rendition matches or no font was found after the callback, then the segment will not be rendered and a warning will be issued.

Render tables are specified in resource files with the following syntax:

resource_spec: [*tag* [, *tag*]*]

where *tag* is some string suitable for the **XmNtag** resource of a rendition.

If no tags are specified, then a render table will be created that is either empty or contains only a rendition with a tag of **_MOTIF_DEFAULT_LOCALE**.

Specific values for specific rendition resources are specified using the following syntax:

resource_spec [*|.] *rendition*[*|.] *resource_name*: *value*

where:

resource_spec
Specifies the render table.

rendition Is either the class Rendition or a tag.

resource_name
Is either the class or name of a particular resource.

value Is the specification of the value to be set.

Any resource line that consists of just a resource name or class component with no rendition component or loose binding will be assumed to specify resource values for a rendition with a tag of **_MOTIF_DEFAULT_LOCALE**. In effect, this creates a default rendition in much the same way that specifying no fontlist tag for a fontlist resource causes the fontlist created to contain an entry tagged with **XmFONTLIST_DEFAULT_TAG**:

resource_spec.*resource_name*: *value*

For example, the following would set the **XmNrenderTable** resource of **label1** to a render table consisting of three renditions tagged with **_MOTIF_DEFAULT_LOCALE**, *bold*, and *oblique*, with values for resources set as described in the resource specifications.

```
*label1.renderTable: bold, oblique
*label1.renderTable.bold.renditionForeground: Green
*label1.renderTable.bold.fontName: *-*-*-bold-*-iso8859-1
*label1.renderTable.bold.fontType: FONT_IS_FONT
*label1.renderTable.oblique.renditionBackground: Red
*label1.renderTable.oblique.fontName: *-*-*-italic-*-iso8859-1
*label1.renderTable.oblique.fontType: FONT_IS_FONT
*label1.renderTable.oblique.underlineType: AS_IS
*label1.renderTable.fontName: fixed
*label1.renderTable.fontType: FONT_IS_FONT
*label1.renderTable.renditionForegound: black
*label1.renderTable*tabList: 1in, +1.5in, +3in
```

Related Information

XmRenderTableAddRenditions(3), **XmRenderTableCopy**(3), **XmRenderTableCvtFromProp**(3), **XmRenderTableCvtToProp**(3), **XmRenderTableFree**(3), **XmRenderTableGetRendition**(3), **XmRenderTableGetRenditions**(3), **XmRenderTableGetTags**(3), **XmRenderTableRemoveRenditions**(3), **XmRendition**(3), and **XmString**(3).

XmString

Purpose

Data type for a compound string

Synopsis

#include <Xm/Xm.h>

Description

XmString is the data type for a compound string. Compound strings consist of a sequence of components, including, but not limited to, the following:

- **XmSTRING_COMPONENT_SEPARATOR**
- **XmSTRING_COMPONENT_TAB**
- **XmSTRING_COMPONENT_LAYOUT_POP**
- **XmSTRING_COMPONENT_DIRECTION**
- **XmSTRING_COMPONENT_LAYOUT_PUSH**
- **XmSTRING_COMPONENT_CHARSET**
- **XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG**
- **XmSTRING_COMPONENT_LOCALE**
- **XmSTRING_COMPONENT_LOCALE_TEXT**
- **XmSTRING_COMPONENT_TAG**
- **XmSTRING_COMPONENT_TEXT**
- **XmSTRING_COMPONENT_END**
- **XmSTRING_COMPONENT_RENDITION_BEGIN**
- **XmSTRING_COMPONENT_RENDITION_END**
- **XmSTRING_COMPONENT_UNKNOWN**
- **XmSTRING_COMPONENT_WIDECHAR_TEXT**

and also a rendition tags table, text, and text component. When a compound string is displayed, the rendition tags and the direction are used to determine how to display the text.

Calling **XtGetValues** for a resource whose type is **XmString** yields a copy of the compound string resource value. The application is responsible for using **XmStringFree** to free the memory allocated for the copy.

Please see the **XmStringComponentType** reference page for more detail about compound string components, and for a description of the order in which the components should appear in a compound string. Refer to the **XmRenderTable** reference page for a description of the algorithm that associates the rendition tags used for displaying a compound string text component with a rendition in a render table.

Related Information

XmParseMapping(3), **XmParseMappingCreate**(3), **XmParseMappingFree**(3), **XmParseMappingGetValues**(3), **XmParseMappingSetValues**(3), **XmParseTable**(3), **XmParseTableFree**(3), **XmStringBaseline**(3), **XmStringByteCompare**(3), **XmStringByteStreamLength**(3), **XmStringCompare**(3), **XmStringComponentCreate**(3), **XmStringComponentType**(3), **XmStringConcat**(3), **XmStringConcatAndFree**(3), **XmStringCopy**(3), **XmStringCreate**(3), **XmStringCreateLocalized**(3), **XmStringCreateLtoR**(3), **XmStringCreateSimple**(3), **XmStringDirection**(3), **XmStringDirectionCreate**(3), **XmStringDirectionToDirection**(3), **XmStringDraw**(3), **XmStringDrawImage**(3), **XmStringDrawUnderline**(3), **XmStringEmpty**(3), **XmStringExtent**(3), **XmStringFree**(3), **XmStringFreeContext**(3), **XmStringGenerate**(3), **XmStringGetLtoR**(3), **XmStringGetNextComponent**(3), **XmStringGetNextSegment**(3), **XmStringGetNextTriple**(3), **XmStringHasSubstring**(3), **XmStringHeight**(3), **XmStringInitContext**(3), **XmStringIsVoid**(3), **XmStringLength**(3), **XmStringLineCount**(3), **XmStringNConcat**(3), **XmStringNCopy**(3), **XmStringParseText**(3), **XmStringPeekNextComponent**(3), **XmStringPeekNextTriple**(3), **XmStringPutRendition**(3), **XmStringSegmentCreate**(3), **XmStringSeparatorCreate**(3), **XmStringTable**(3), **XmStringTableParseStringArray**(3), **XmStringTableProposeTablist**(3), **XmStringTableToXmString**(3), **XmStringTableUnparse**(3), **XmStringToXmStringTable**(3), **XmStringUnparse**(3), **XmStringWidth**(3),

XmCvtXmStringToByteStream(3), **XmCvtXmStringToCT**(3),
XmCvtCTToXmString(3), and **XmCvtByteStreamToXmString**(3).

XmStringDirection

Purpose

Data type for the direction of display in a string

Synopsis

#include <Xm/Xm.h>

Description

XmStringDirection is the data type for specifying the direction in which the system displays characters of a string, or characters of a segment of a compound string. This is an enumeration with three possible values:

XmSTRING_DIRECTION_L_TO_R
Specifies left to right display

XmSTRING_DIRECTION_R_TO_L
Specifies right to left display

XmSTRING_DIRECTION_DEFAULT
Specifies that the display direction will be set by the widget in which the compound string is to be displayed.

Related Information

XmString(3).

XmStringTable

Purpose

Data type for an array of compound strings

Synopsis

#include <Xm/Xm.h>

Description

XmStringTable is the data type for an array of compound strings (objects of type **XmString**).

Related Information

XmString(3).

XmTab

Purpose

Data type for a tab stop

Synopsis

#include <Xm/Xm.h>
XmTab

Description

XmTab is a data structure that specifies a tab stop to be used in rendering an **XmString** containing tab components. An **XmTab** value contains a value, a unit type, an offset model (either **XmABSOLUTE** or **XmRELATIVE**), an alignment model (**XmALIGNMENT_BEGINNING**), and a decimal point character. The resource file syntax for **XmTab** is specified in the **XmTabList** reference page.

Related Information

XmTabCreate(3), **XmTabFree**(3), **XmTabGetValues**(3), **XmTabList**(3), and **XmTabSetValue**(3).

XmTabList

Purpose

Data type for a tab list

Synopsis

#include <Xm/Xm.h>
XmTabList

Description

XmTabList is the data type for a tab list. A tab list consists of tab stop list entries (**XmTab**s). Whenever a tab component is encountered while an **XmString** is being rendered, the origin of the next X draw depends on the next **XmTab**. If a tab stop would cause text to overlap, the x position for the segment is reset to follow immediately after the end of the previous segment.

Tab lists are specified in resource files with the following syntax:

resource_spec: *tab* WHITESPACE [, WHITESPACE *tab*]*

The resource value string consists of one or more tabs separated by commas. Each *tab* identifies the value of the tab, the unit type, and whether the offset is relative or absolute. For example:

tab := *float* [WHITESPACE *units*]
float := [*sign*] [[DIGIT]*.]DIGIT+
sign := +

where the presence or absence of *sign* indicates, respectively, a relative offset or an absolute offset. Note that negative tab values are not allowed. *units* indicates the unitType to use as described in the **XmConvertUnits** reference page.

For example, the following specifies a tab list consisting of a one inch absolute tab followed by a one inch relative tab:

```
*tabList: 1in, +1in
```

For resources of type, dimension, or position, you can specify units as described in the **XmNunitType** resource of the **XmGadget**, **XmManager**, or **XmPrimitive** reference page.

Related Information

Refer to the *Motif 2.1—Programmer’s Guide* for more information about tabs and tab lists. **XmTabListCopy**(3), **XmTabListFree**(3), **XmTabListGetTab**(3), **XmTabListInsertTabs**(3), **XmTabListRemoveTabs**(3), **XmTabListReplacePositions**(3), and **XmTabListTabCount**(3).

XmTextPosition

Purpose

Data type for a character position within a text string

Synopsis

#include <Xm/Xm.h>

Description

XmTextPosition is an integer data type that holds a character's position within a text string for Text and TextField.

An **XmTextPosition** value conceptually points to the gap between two characters. For example, consider a text string consisting of **N** characters. A value of 0 refers to the position immediately prior to the first character. A value of 1 refers to the position in between the first and second characters. A value of **N** refers to the position immediately following the last character. Therefore, the text string of **N** characters actually contains **N + 1** positions.

Related Information

XmText(3).

Chapter 6

Xm Functions

XmActivateProtocol

Purpose

A VendorShell function that activates a protocol

Synopsis

#include <Xm/Protocols.h>

void XmActivateProtocol(
Widget *shell***,**
Atom *property***,**
Atom *protocol***);**

Description

XmActivateProtocol activates a protocol. It updates the handlers and the *property* if the *shell* is realized. It is sometimes useful to allow a protocol's state information (callback lists, and so on) to persist, even though the client may choose to temporarily resign from the interaction. This is supported by allowing a *protocol* to be in one of two states: active or inactive. If the *protocol* is active and the *shell* is realized, the *property* contains the *protocol* **Atom**. If the *protocol* is inactive, the **Atom** is not present in the *property*.

XmActivateWMProtocol is a convenience interface. It calls **XmActivateProtocol** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocol	Specifies the protocol **Atom**

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmActivateWMProtocol**(3), **XmRemoveProtocols**(3) and **XmInternAtom**(3).

XmActivateWMProtocol

Purpose

A VendorShell convenience interface that activates a protocol

Synopsis

```
#include <Xm/Protocols.h>

void XmActivateWMProtocol(
        Widget shell,
        Atom protocol);
```

Description

XmActivateWMProtocol is a convenience interface. It calls **XmActivateProtocol** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
protocol	Specifies the protocol **Atom**

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmActivateProtocol**(3), **XmInternAtom**(3), and **XmRemoveWMProtocols**(3).

XmAddProtocolCallback

Purpose

A VendorShell function that adds client callbacks for a protocol

Synopsis

#include <Xm/Protocols.h>

void XmAddProtocolCallback(
    Widget *shell*,
    Atom *property*,
    Atom *protocol*,
    XtCallbackProc *callback*,
    XtPointer *closure*);

Description

XmAddProtocolCallback adds client callbacks for a protocol. It checks if the protocol is registered, and if it is not, calls **XmAddProtocols**. It then adds the callback to the internal list. These callbacks are called when the corresponding client message is received.

XmAddWMProtocolCallback is a convenience interface. It calls **XmAddProtocolCallback** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocol	Specifies the protocol **Atom**
callback	Specifies the procedure to call when a protocol message is received
closure	Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddWMProtocolCallback**(3), **XmInternAtom**(3), and **XmRemoveProtocolCallback**(3).

XmAddProtocols

Purpose

A VendorShell function that adds the protocols to the protocol manager and allocates the internal tables

Synopsis

#include <Xm/Protocols.h>

void XmAddProtocols(
    Widget *shell***,**
    Atom *property***,**
    Atom ****protocols*,**
    Cardinal *num_protocols***);**

Description

XmAddProtocols adds the protocols to the protocol manager and allocates the internal tables.

XmAddWMProtocols is a convenience interface. It calls **XmAddProtocols** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocols	Specifies the protocol **Atoms**
num_protocols	Specifies the number of elements in *protocols*

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddWMProtocols**(3), **XmInternAtom**(3), and **XmRemoveProtocols**(3).

XmAddTabGroup

Purpose

A function that adds a manager or a primitive widget to the list of tab groups

Synopsis

#include <Xm/Xm.h>

void XmAddTabGroup(
Widget *tab_group***);**

Description

This function is obsolete and its behavior is replaced by setting **XmNnavigationType** to **XmEXCLUSIVE_TAB_GROUP**. When the keyboard is used to traverse through a widget hierarchy, primitive or manager widgets are grouped together into what are known as **tab groups**. Any manager or primitive widget can be a tab group. Within a tab group, move the focus to the next widget in the tab group by using the arrow keys. To move to another tab group, use **KNextField** or **KPrevField**.

Tab groups are ordinarily specified by the **XmNnavigationType** resource. **XmAddTabGroup** is called to control the order of traversal of tab groups. The widget specified by *tab_group* is appended to the list of tab groups to be traversed, and the widget's **XmNnavigationType** is set to **XmEXCLUSIVE_TAB_GROUP**.

tab_group	Specifies the manager or primitive widget ID

Related Information

XmManager(3), **XmGetTabGroup**(3), **XmPrimitive**(3), and **XmRemoveTabGroup**(3).

XmAddToPostFromList

Purpose

a RowColumn function that makes a menu accessible from more than one widget

Synopsis

#include <Xm/RowColumn.h>

void XmAddToPostFromList(
    Widget *menu***,**
    Widget *post_from_widget***);**

Description

XmAddToPostFromList makes a menu accessible from more than one widget. After a menu is once created, this function may be used to make that menu accessible from a second widget. The process may be repeated indefinitely. In other words, where an application would use **XmCreatePopupMenu** or **XmCreatePulldownMenu** or their equivalent to create a new menu identical to one that already exists, it can use this function to reuse that earlier menu.

If *menu* refers to a Popup menu, then the *post_from_widget* widget can now pop up the specified menu. The actual posting of the menu occurs as it always does, either through an event handler, or the automatic popup menu support (see the **XmRowColumn**(3) reference page).

If *menu* refers to a Pulldown menu, its ID is placed in the **XmNsubMenuId** resource of the specified *post_from_widget*. In this case, the *post_from_widget* widget must be either a CascadeButton or a CascadeButtonGadget.

Note that this function manipulates the internal structures themselves, not a copy of them.

menu	Specifies the ID of the RowColumn widget containing the menu (Popup or Pulldown) to be made accessible from the widget.

post_from_widget
Specifies the widget ID of the widget which will now be able to post the menu specified by *menu*.

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Related Information

XmGetPostedFromWidget(3), **XmRemoveFromPostFromList**(3), and **XmRowColumn**(3).

XmAddWMProtocolCallback

Purpose

A VendorShell convenience interface that adds client callbacks for a protocol

Synopsis

#include <Xm/Protocols.h>

void XmAddWMProtocolCallback(
  Widget *shell***,**
  Atom *protocol***,**
  XtCallbackProc *callback***,**
  XtPointer *closure***);**

Description

XmAddWMProtocolCallback is a convenience interface. It calls **XmAddProtocolCallback** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
protocol	Specifies the protocol **Atom**
callback	Specifies the procedure to call when a protocol message is received
closure	Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddProtocolCallback**(3), **XmInternAtom**(3), and **XmRemoveWMProtocolCallback**(3).

XmAddWMProtocols

Purpose

A VendorShell convenience interface that adds the protocols to the protocol manager and allocates the internal tables

Synopsis

#include <Xm/Protocols.h>

void XmAddWMProtocols(
    Widget *shell***,**
    Atom ****protocols*,**
    Cardinal *num_protocols***);**

Description

XmAddWMProtocols is a convenience interface. It calls **XmAddProtocols** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated

protocols Specifies the protocol **Atoms**

num_protocols
Specifies the number of elements in *protocols*

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddProtocols**(3), **XmInternAtom**(3), and **XmRemoveWMProtocols**.

XmCascadeButtonGadgetHighlight

Purpose

A CascadeButtonGadget function that sets the highlight state

Synopsis

#include <Xm/CascadeBG.h>

void XmCascadeButtonGadgetHighlight(
Widget *cascadeButtonGadget***,**
Boolean *highlight***);**

Description

XmCascadeButtonGadgetHighlight either draws or erases the shadow highlight around the CascadeButtonGadget.

cascadeButtonGadget
Specifies the CascadeButtonGadget to be highlighted or unhighlighted

highlight Specifies whether to highlight (True) or to unhighlight (False)

For a complete definition of CascadeButtonGadget and its associated resources, see **XmCascadeButtonGadget**(3).

Related Information

XmCascadeButton(3), **XmCascadeButtonGadget**(3), and **XmCascadeButtonHighlight**(3).

XmCascadeButtonHighlight

Purpose

A CascadeButton and CascadeButtonGadget function that sets the highlight state

Synopsis

#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>

void XmCascadeButtonHighlight(
        Widget *cascadeButton***,**
        Boolean *highlight***);**

Description

XmCascadeButtonHighlight either draws or erases the shadow highlight around the CascadeButton or the CascadeButtonGadget.

cascadeButton
: Specifies the CascadeButton or CascadeButtonGadget to be highlighted or unhighlighted

highlight
: Specifies whether to highlight (True) or to unhighlight (False)

For a complete definition of CascadeButton or CascadeButtonGadget and their associated resources, see **XmCascadeButton**(3) or **XmCascadeButtonGadget**(3).

Related Information

XmCascadeButton(3), **XmCascadeButtonGadget**(3) and **XmCascadeButtonGadgetHighlight**(3).

XmChangeColor

Purpose

Recalculates all associated colors of a widget

Synopsis

#include <Xm/Xm.h>

void XmChangeColor(
    Widget *widget***,**
    Pixel *background***);**

Description

XmChangeColor handles all color modifications for the specified widget when a new background pixel value is specified. This function recalculates the foreground, select, and shadow colors based on the new background color and sets the corresponding resources for the widget. If a color calculation procedure has been set by a call to **XmSetColorCalculation**, **XmChangeColor** uses that procedure to calculate the new colors. Otherwise, the routine uses a default procedure.

widget	Specifies the widget ID whose colors will be updated
background	Specifies the background color pixel value

Related Information

XmGetColorCalculation(3), **XmGetColors**(3), and **XmSetColorCalculation**(3).

XmClipboardCancelCopy

Purpose

A clipboard function that cancels a copy to the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardCancelCopy (*display, window, item_id***)**
    Display * *display***;**
    Window *window***;**
    long *item_id***;**

Description

XmClipboardCancelCopy cancels the copy to clipboard that is in progress and frees up temporary storage. When a copy is to be performed, **XmClipboardStartCopy** allocates temporary storage for the clipboard data. **XmClipboardCopy** copies the appropriate data into the the temporary storage. **XmClipboardEndCopy** copies the data to the clipboard structure and frees up the temporary storage structures. If **XmClipboardCancelCopy** is called, the **XmClipboardEndCopy** function does not have to be called. A call to **XmClipboardCancelCopy** is valid only after a call to **XmClipboardStartCopy**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies a widget's window ID that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to **XmClipboardStartCopy**.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardFail
The function failed because **XmClipboardStartCopy** was not called or because the data item contains too many formats.

Related Information

XmClipboardCopy(3), **XmClipboardEndCopy**(3), and **XmClipboardStartCopy**(3).

XmClipboardCopy

Purpose

A clipboard function that copies a data item to temporary storage for later copying to clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardCopy (*display, window, item_id, format_name,*
*buffer, length, private_id, data_id***)**
Display * *display***;**
Window *window***;**
long *item_id***;**
char * *format_name***;**
XtPointer *buffer***;**
unsigned long *length***;**
long *private_id***;**
long * *data_id***;**

Description

XmClipboardCopy copies a data item to temporary storage. The data item is moved from temporary storage to the clipboard data structure when a call to **XmClipboardEndCopy** is made. Additional calls to **XmClipboardCopy** before a call to **XmClipboardEndCopy** add additional data item formats to the same data item or append data to an existing format. Formats are described in the *Inter-Client Communication Conventions Manual* (ICCCM) as targets.

NOTE: Do not call **XmClipboardCopy** before a call to **XmClipboardStartCopy** has been made. The latter function allocates temporary storage required by **XmClipboardCopy**.

If the *buffer* argument is NULL, the data is considered to be passed by name. When data that has been passed by name is later requested by another application, the application that owns the data receives a callback with a request for the data. The application that owns the data must then transfer the data to the clipboard with the **XmClipboardCopyByName** function. When a data item that was passed by name is deleted from the clipboard, the application that owns the data receives a callback stating that the data is no longer needed.

For information on the callback function, see the callback argument description for **XmClipboardStartCopy**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to **XmClipboardStartCopy**.

format_name
Specifies the name of the format in which the data item is stored on the clipboard. The format was known as target in the ICCCM.

buffer Specifies the buffer from which the clipboard copies the data.

length Specifies the length, in bytes, of the data being copied to the clipboard.

private_id Specifies the private data that the application wants to store with the data item.

data_id Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This argument is required only for data that is passed by name.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardFail
The function failed because **XmClipboardStartCopy** was not called or because the data item contains too many formats.

Related Information

XmClipboardCopyByName(3), **XmClipboardEndCopy**(3), and **XmClipboardStartCopy**(3).

XmClipboardCopyByName

Purpose

A clipboard function that copies a data item passed by name

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardCopyByName (*display, window, data_id,*
*buffer, length, private_id***)**
Display * *display***;**
Window *window***;**
long *data_id***;**
XtPointer *buffer***;**
unsigned long *length***;**
long *private_id***;**

Description

XmClipboardCopyByName copies the actual data for a data item that was previously passed by name to the clipboard. Data is considered to be passed by name when a call to **XmClipboardCopy** is made with a NULL buffer parameter. Additional calls to this function append new data to the existing data.

display	Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.
window	Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.
data_id	Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This number was assigned by **XmClipboardCopy** to the data item.
buffer	Specifies the buffer from which the clipboard copies the data.
length	Specifies the number of bytes in the data item.

private_id Specifies the private data that the application wants to store with the data item.

Return Values

XmClipboardSuccess
: The function was successful.

XmClipboardLocked
: The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardCopy(3), **XmClipboardLock**(3), **XmClipboardStartCopy**(3), and **XmClipboardUnlock**(3).

XmClipboardEndCopy

Purpose

A clipboard function that completes the copying of data to the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardEndCopy (*display, window, item_id***)**
    Display * *display***;**
    Window *window***;**
    long *item_id***;**

Description

XmClipboardEndCopy locks the clipboard from access by other applications, places data in the clipboard data structure, and unlocks the clipboard. Data items copied to the clipboard by **XmClipboardCopy** are not actually entered in the clipboard data structure until the call to **XmClipboardEndCopy**.

This function also frees up temporary storage that was allocated by **XmClipboardStartCopy**, which must be called before **XmClipboardEndCopy**. The latter function should not be called if **XmClipboardCancelCopy** has been called.

display	Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.
window	Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.
item_id	Specifies the number assigned to this data item, which was returned by a previous call to **XmClipboardStartCopy**.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardFail
The function failed because **XmClipboardStartCopy** was not called.

Related Information

XmClipboardCancelCopy(3), **XmClipboardCopy**(3) and **XmClipboardStartCopy**(3).

XmClipboardEndRetrieve

Purpose

A clipboard function that completes retrieval of data from the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardEndRetrieve (*display, window***)**
        Display * *display***;**
        Window *window***;**

Description

XmClipboardEndRetrieve suspends copying data incrementally from the clipboard. It tells the clipboard routines that the application is through copying an item from the clipboard. Until this function is called, data items can be retrieved incrementally from the clipboard with **XmClipboardRetrieve**. The act of copying data is started with the **XmClipboardStartRetrieve** function.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained with **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application

the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardRetrieve(3), **XmClipboardStartCopy**(3), and **XmClipboardStartRetrieve**(3).

XmClipboardInquireCount

Purpose

A clipboard function that returns the number of data item formats

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardInquireCount (*display, window, count,*
*max_format_name_length***)**
Display * *display***;**
Window *window***;**
int * *count***;**
unsigned long * *max_format_name_length***;**

Description

XmClipboardInquireCount returns the number of data item formats available for the data item in the clipboard. This function also returns the maximum name-length for all formats in which the data item is stored.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

count Returns the number of data item formats available for the data item in the clipboard. If no formats are available, this argument equals 0 (zero). The count includes the formats that were passed by name.

max_format_name_length
Specifies the maximum length of all format names for the data item in the clipboard.

Return Values

XmClipboardSuccess
: The function was successful.

XmClipboardLocked
: The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardNoData
: The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardStartCopy(3).

XmClipboardInquireFormat

Purpose

A clipboard function that returns a specified format name

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardInquireFormat (*display, window, index, format_name_buf, buffer_len, copied_len***)**
  Display * *display***;**
  Window *window***;**
  int *index***;**
  XtPointer *format_name_buf***;**
  unsigned long *buffer_len***;**
  unsigned long * *copied_len***;**

Description

XmClipboardInquireFormat returns a specified format name for the data item in the clipboard. If the name must be truncated, the function returns a warning status.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

index Specifies which of the ordered format names to obtain. If this index is greater than the number of formats for the data item, this function returns a 0 (zero) in the *copied_len* argument.

format_name_buf
Specifies the buffer that receives the format name.

buffer_len Specifies the number of bytes in the format name buffer.

copied_len Specifies the number of bytes in the data item copied to the buffer. If this argument equals 0 (zero), there is no *n*th format for the data item.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardTruncate
The data returned is truncated because the user did not provide a buffer large enough to hold the data.

XmClipboardNoData
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardStartCopy(3).

XmClipboardInquireLength

Purpose

A clipboard function that returns the length of the stored data

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardInquireLength (*display, window, format_name, length***)**
    Display * *display***;**
    Window *window***;**
    char * *format_name***;**
    unsigned long * *length***;**

Description

XmClipboardInquireLength returns the length of the data stored under a specified format name for the clipboard data item. If no data is found for the specified format, or if there is no item on the clipboard, this function returns a value of 0 (zero) in the *length* argument.

Any format passed by name is assumed to have *length* passed in a call to **XmClipboardCopy**, even though the data has not yet been transferred to the clipboard in that format.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name
Specifies the name of the format for the data item.

length Specifies the length of the next data item in the specified format. This argument equals 0 (zero) if no data is found for the specified format, or if there is no item on the clipboard.

Return Values

XmClipboardSuccess
: The function was successful.

XmClipboardLocked
: The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardNoData
: The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardCopy(3) and **XmClipboardStartCopy**(3).

XmClipboardInquirePendingItems

Purpose

A clipboard function that returns a list of data ID/private ID pairs

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardInquirePendingItems (*display, window, format_name, item_list, count***)**
  Display * *display***;**
  Window *window***;**
  char * *format_name***;**
  XmClipboardPendingList * *item_list***;**
  unsigned long * *count***;**

Description

XmClipboardInquirePendingItems returns a list of data ID/private ID pairs for the specified format name. A data item is considered pending if the application originally passed it by name, the application has not yet copied the data, and the item has not been deleted from the clipboard. The application is responsible for freeing the memory provided by this function to store the list. To free the memory, call **XtFree**.

This function is used by an application when exiting, to determine if the data that is passed by name should be sent to the clipboard.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name
Specifies a string that contains the name of the format for which the list of data ID/private ID pairs is to be obtained.

item_list Specifies the address of the array of data ID/private ID pairs for the specified format name. This argument is a type

XmClipboardPendingList. The application is responsible for freeing the memory provided by this function for storing the list.

count Specifies the number of items returned in the list. If there is no data for the specified format name, or if there is no item on the clipboard, this argument equals 0 (zero).

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardStartCopy(3).

XmClipboardLock

Purpose

A clipboard function that locks the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardLock (*display, window***)**
    Display * *display***;**
    Window *window***;**

Description

XmClipboardLock locks the clipboard from access by another application until **XmClipboardUnlock** is called. All clipboard functions lock and unlock the clipboard to prevent simultaneous access. This function allows the application to keep the clipboard data from changing between calls to **Inquire** and other clipboard functions. The application does not need to lock the clipboard between calls to **XmClipboardStartCopy** and **XmClipboardEndCopy** or to **XmClipboardStartRetrieve** and **XmClipboardEndRetrieve**.

If the clipboard is already locked by another application, **XmClipboardLock** returns an error status. Multiple calls to this function by the same application increase the lock level.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardEndCopy(3), **XmClipboardEndRetrieve**(3), **XmClipboardStartCopy**(3), **XmClipboardStartRetrieve**(3), and **XmClipboardUnlock**(3).

XmClipboardRegisterFormat

Purpose

A clipboard function that registers a new format

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardRegisterFormat (*display*, *format_name*, *format_length***)**
    Display * *display***;**
    char * *format_name***;**
    int *format_length***;**

Description

XmClipboardRegisterFormat registers a new format. Each format stored on the clipboard should have a length associated with it; this length must be known to the clipboard routines. Formats are known as targets in the *Inter-Client Communication Conventions Manual* (ICCCM). All of the formats specified by version 1.1 of the ICCCM conventions are preregistered. Any other format that the application wants to use must either be 8-bit data or be registered via this routine. Failure to register the length of the data results in incompatible applications across platforms having different byte-swapping orders.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

format_name
Specifies the string name for the new format (target).

format_length
Specifies the format length in bits (8, 16, or 32).

Return Values

XmClipboardBadFormat
The *format_name* must not be NULL, and the *format_length* must be 8, 16, or 32.

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardFail
The function failed because the specified format was already registered with a different length from that specified now. If a specified format was already registered with the same length as that specified now, *XmClipboardSuccess* is returned.

Related Information

XmClipboardStartCopy(3).

XmClipboardRetrieve

Purpose

A clipboard function that retrieves a data item from the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardRetrieve (*display, window, format_name,*
*buffer, length, num_bytes, private_id***)**
Display * *display***;**
Window *window***;**
char * *format_name***;**
XtPointer *buffer***;**
unsigned long *length***;**
unsigned long * *num_bytes***;**
long * *private_id***;**

Description

XmClipboardRetrieve retrieves the current data item from clipboard storage. It returns a warning if the clipboard is locked, if there is no data on the clipboard, or if the data needs to be truncated because the buffer length is too short.

Between a call to **XmClipboardStartRetrieve** and a call to **XmClipboardEndRetrieve**, multiple calls to **XmClipboardRetrieve** with the same format name result in data being incrementally copied from the clipboard until the data in that format has all been copied.

The return value *XmClipboardTruncate* from calls to **XmClipboardRetrieve** indicates that more data remains to be copied in the given format. It is recommended that any calls to the **Inquire** functions that the application needs to make to effect the copy from the clipboard be made between the call to **XmClipboardStartRetrieve** and the first call to **XmClipboardRetrieve**. This way, the application does not need to call **XmClipboardLock** and **XmClipboardUnlock**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name
Specifies the name of a format in which the data is stored on the clipboard.

buffer Specifies the buffer to which the application wants the clipboard to copy the data. The function allocates space to hold the data returned into the buffer. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XtFree**.

length Specifies the length of the application buffer.

num_bytes Specifies the number of bytes of data copied into the application buffer.

private_id Specifies the private data stored with the data item by the application that placed the data item on the clipboard. If the application did not store private data with the data item, this argument returns 0 (zero).

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardTruncate
The data returned is truncated because the user did not provide a buffer large enough to hold the data.

XmClipboardNoData
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardEndRetrieve(3), **XmClipboardLock**(3), **XmClipboardStartCopy**(3), **XmClipboardStartRetrieve**(3), and **XmClipboardUnlock**(3).

XmClipboardStartCopy

Purpose

A clipboard function that sets up a storage and data structure

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardStartCopy (*display, window, clip_label,*
*timestamp, widget, callback, item_id***)**
Display * *display***;**
Window *window***;**
XmString *clip_label***;**
Time *timestamp***;**
Widget *widget***;**
XmCutPasteProc *callback***;**
long * *item_id***;**

Description

XmClipboardStartCopy sets up storage and data structures to receive clipboard data. An application calls this function during a cut or copy operation. The data item that these structures receive then becomes the next data item in the clipboard.

Copying a large piece of data to the clipboard can take a long time. It is possible that, once the data is copied, no application will ever request that data. The Motif Toolkit provides a mechanism so that an application does not need to actually pass data to the clipboard until the data has been requested by some application.

Instead, the application passes format and length information in **XmClipboardCopy** to the clipboard functions, along with a widget ID and a callback function address that is passed in **XmClipboardStartCopy**. The widget ID is necessary for communications between the clipboard functions in the application that owns the data and the clipboard functions in the application that requests the data.

The callback functions are responsible for copying the actual data to the clipboard through **XmClipboardCopyByName**. The callback function is also called if the data item is removed from the clipboard and the actual data is no longer needed.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

clip_label Specifies the label to be associated with the data item. This argument is used to identify the data item, as in a clipboard viewer. An example of a label is the name of the application that places the data in the clipboard.

timestamp Specifies the time of the event that triggered the copy. A valid timestamp must be supplied; it is not sufficient to use **CurrentTime**.

widget Specifies the ID of the widget that receives messages requesting data previously passed by name. This argument must be present in order to pass data by name. Any valid widget ID in your application can be used for this purpose and all the message handling is taken care of by the cut and paste functions.

callback Specifies the address of the callback function that is called when the clipboard needs data that was originally passed by name. This is also the callback to receive the *delete* message for items that were originally passed by name. This argument must be present in order to pass data by name.

item_id Specifies the number assigned to this data item. The application uses this number in calls to **XmClipboardCopy**, **XmClipboardEndCopy**, and **XmClipboardCancelCopy**.

For more information on passing data by name, see **XmClipboardCopy**(3) and **XmClipboardCopyByName**(3).

The *widget* and *callback* arguments must be present in order to pass data by name. The callback format is as follows:

```
void (*callback) (widget, data_id, private, reason)
        Widget  widget;
        long    *data_id;
        long    *private;
        int     *reason;
```

widget Specifies the ID of the widget passed to this function.

data_id	Specifies the identifying number returned by **XmClipboardCopy**, which identifies the pass-by-name data.
private	Specifies the private information passed to **XmClipboardCopy**.
reason	Specifies the reason. **XmCR_CLIPBOARD_DATA_DELETE** or **XmCR_CLIPBOARD_DATA_REQUEST** are the possible values.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardCancelCopy(3), **XmClipboardCopy**(3), **XmClipboardCopyByName**(3), **XmClipboardEndCopy**(3), **XmClipboardEndRetrieve**(3), **XmClipboardInquireCount**(3), **XmClipboardInquireFormat**(3), **XmClipboardInquireLength**(3), **XmClipboardInquirePendingItems**(3), **XmClipboardLock**(3), **XmClipboardRegisterFormat**(3), **XmClipboardRetrieve**(3), **XmClipboardStartRetrieve**(3), **XmClipboardUndoCopy**(3), **XmClipboardUnlock**(3), and **XmClipboardWithdrawFormat**(3).

XmClipboardStartRetrieve

Purpose

A clipboard function that prepares to retrieve data from the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardStartRetrieve (*display, window, timestamp***)**
    Display * *display***;**
    Window *window***;**
    Time *timestamp***;**

Description

XmClipboardStartRetrieve tells the clipboard routines that the application is ready to start copying an item from the clipboard. The clipboard is locked by this routine and stays locked until **XmClipboardEndRetrieve** is called. Between a call to **XmClipboardStartRetrieve** and a call to **XmClipboardEndRetrieve**, multiple calls to **XmClipboardRetrieve** with the same format name result in data being incrementally copied from the clipboard until the data in that format has all been retrieved.

A return value of *XmClipboardTruncate* from calls to **XmClipboardRetrieve** indicates that more data remains to be copied in the given format. It is recommended that any calls to the **Inquire** functions that the application needs to make to complete the copy from the clipboard be made between the call to **XmClipboardStartRetrieve** and the first call to **XmClipboardRetrieve**. This way, the application does not need to call **XmClipboardLock** and **XmClipboardUnlock**.

display	Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.
window	Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.
timestamp	Specifies the time of the event that triggered the copy. A valid timestamp must be supplied; it is not sufficient to use **CurrentTime**.

Return Values

XmClipboardSuccess
: The function is successful.

XmClipboardLocked
: The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardEndRetrieve(3), **XmClipboardInquireCount**(3), **XmClipboardInquireFormat**(3), **XmClipboardInquireLength**(3), **XmClipboardInquirePendingItems**(3), **XmClipboardLock**(3), **XmClipboardRetrieve**(3), **XmClipboardStartCopy**(3), and **XmClipboardUnlock**(3).

XmClipboardUndoCopy

Purpose

A clipboard function that deletes the last item placed on the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardUndoCopy (*display*, *window***)**
    Display * *display***;**
    Window *window***;**

Description

XmClipboardUndoCopy deletes the last item placed on the clipboard if the item was placed there by an application with the passed *display* and *window* arguments. Any data item deleted from the clipboard by the original call to **XmClipboardCopy** is restored. If the *display* or *window* IDs do not match the last copied item, no action is taken, and this function has no effect.

display	Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.
window	Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application

the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardLock(3) and **XmClipboardStartCopy**(3).

XmClipboardUnlock

Purpose

A clipboard function that unlocks the clipboard

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardUnlock (*display*, *window*, *remove_all_locks***)**
    Display * *display***;**
    Window *window***;**
    Boolean *remove_all_locks***;**

Description

XmClipboardUnlock unlocks the clipboard, enabling it to be accessed by other applications.

If multiple calls to **XmClipboardLock** have occurred, the same number of calls to **XmClipboardUnlock** is necessary to unlock the clipboard, unless *remove_all_locks* is set to True.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

remove_all_locks
When True, indicates that all nested locks should be removed. When False, indicates that only one level of lock should be removed.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardFail
The function failed because the clipboard was not locked or was locked by another application.

Related Information

XmClipboardCancelCopy(3), **XmClipboardCopy**(3), **XmClipboardEndCopy**(3), **XmClipboardEndRetrieve**(3), **XmClipboardInquireCount**(3), **XmClipboardInquireFormat**(3), **XmClipboardInquireLength**(3), **XmClipboardInquirePendingItems**(3), **XmClipboardLock**(3), **XmClipboardRegisterFormat**(3), **XmClipboardRetrieve**(3), **XmClipboardStartCopy**(3), **XmClipboardStartRetrieve**(3), **XmClipboardUndoCopy**(3), and **XmClipboardWithdrawFormat**(3).

XmClipboardWithdrawFormat

Purpose

A clipboard function that indicates that the application no longer wants to supply a data item

Synopsis

#include <Xm/CutPaste.h>
int XmClipboardWithdrawFormat (*display, window, data_id***)**
    Display * *display***;**
    Window *window***;**
    long *data_id***;**

Description

XmClipboardWithdrawFormat indicates that the application no longer supplies a data item to the clipboard that the application had previously passed by name.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.

data_id Specifies an identifying number assigned to the data item, that uniquely identifies the data item and the format. This was assigned to the item when it was originally passed by **XmClipboardCopy**.

Return Values

XmClipboardSuccess
The function was successful.

XmClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with

the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardCopy(3) and **XmClipboardStartCopy**(3).

XmComboBoxAddItem

Purpose

add an item to the ComboBox widget

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxAddItem(
    Widget *w***,**
    XmString *item***,**
    int *pos***,**
    Boolean *unique***);**

Description

The **XmComboBoxAddItem** function adds the given item to the XmComboBox at the given position.

The *w* argument specifies the XmComboBox widget ID.

The *item* argument specifies the **XmString** for the new item.

The *pos* argument specifies the position of the new item.

The *unique* argument specifies if this item should duplicate an identical item or not.

Application Usage

The functions **XmComboBoxAddItem** and **XmComboBoxDeletePos** have different naming conventions (Item versus Pos) because of the objects they are manipulating. The Item is a string to be added, the Pos is a numeric position number.

Return Values

The **XmComboBoxAddItem** function returns no value.

Related Information

XmComboBoxDeletePos(3), **XmComboBoxSetItem**(3), **XmComboBoxSelectItem**(3).

XmComboBoxDeletePos

Purpose

Delete a XmComboBox item

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxDeletePos(
    Widget *w***,**
    int *pos***);**

Description

The **XmComboBoxDeletePos** function deletes a specified item from a XmComboBox widget.

The *w* argument specifies the XmComboBox widget ID.

The *pos* argument specifies the position of the item to be deleted.

Application Usage

The functions **XmComboBoxAddItem** and **XmComboBoxDeletePos** have different naming conventions (Item versus Pos) because of the objects they are manipulating. The Item is a string to be added, the Pos is a numeric position number.

Return Values

The **XmComboBoxDeletePos** function returns no value.

Related Information

XmComboBoxAddItem(3), **XmComboBoxSetItem**(3),
XmComboBoxSelectItem(3).

XmComboBoxSelectItem

Purpose

select a XmComboBox item

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxSelectItem(
        Widget *w*,
        XmString *item*);

Description

The **XmComboBoxSelectItem** function selects an item in the XmList of the XmComboBox widget.

The *w* argument specifies the XmComboBox widget ID.

The *item* argument specifies the **XmString** of the item to be selected. If the *item* is not found on the list, **XmComboBoxSelectItem** notifies the user via the **XtWarning** function.

Return Values

The **XmComboBoxSelectItem** function returns no value.

Related Information

XmComboBoxAddItem(3), **XmComboBoxDeletePos**(3), **XmComboBoxSetItem**(3); **XtWarning**(3). in the CAE Specification, Window Management: X Toolkit Intrinsics.

XmComboBoxSetItem

Purpose

set an item in the XmComboBox list

Synopsis

```
#include <Xm/ComboBox.h>

void XmComboBoxSetItem(
        Widget w,
        XmString item);
```

Description

The **XmComboBoxSetItem** function selects an item in the XmList of the given XmComboBox widget and makes it the first visible item in the list.

The *w* argument specifies the XmComboBox widget ID.

The *item* argument specifies the **XmString** for the item to be set in the XmComboBox. If the *item* is not found on the list, **XmComboBoxSetItem** notifies the user via the **XtWarning** function.

Return Values

The **XmComboBoxSetItem** function returns no value.

Related Information

XmComboBoxAddItem(3), **XmComboBoxDeletePos**(3), **XmComboBoxSelectItem**(3); **XtWarning**(3). in the CAE Specification, Window Management: X Toolkit Intrinsics.

XmComboBoxUpdate

Purpose

A ComboBox function that resynchronizes data

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxUpdate(
Widget *widget***);**

Description

XmComboBoxUpdate resynchronizes the internal data structures of a specified ComboBox widget. This function is useful when an application manipulates ComboBox's child widgets, possibly changing data structures. For example, you might want to use the **XmComboBoxUpdate** function after a ComboBox List child selection has been changed without notification.

widget	Specifies the ComboBox widget ID.

Related Information

XmComboBox(3).

XmCommandAppendValue

Purpose

A Command function that appends the passed XmString to the end of the string displayed in the command area of the widget

Synopsis

#include <Xm/Command.h>

void XmCommandAppendValue(
        Widget *widget***,**
        XmString *command***);**

Description

XmCommandAppendValue appends the passed **XmString** to the end of the string displayed in the command area of the Command widget.

widget	Specifies the Command widget ID
command	Specifies the passed **XmString**

For a complete definition of Command and its associated resources, see **XmCommand**(3).

Related Information

XmCommand(3).

XmCommandError

Purpose

A Command function that displays an error message

Synopsis

#include <Xm/Command.h>

void XmCommandError(
Widget *widget*,
XmString *error***);**

Description

XmCommandError displays an error message in the history area of the Command widget. The **XmString** error is displayed until the next command entered occurs.

widget	Specifies the Command widget ID
error	Specifies the passed **XmString**

For a complete definition of Command and its associated resources, see **XmCommand**(3).

Related Information

XmCommand(3).

XmCommandGetChild

Purpose

A Command function that is used to access a component

Synopsis

#include <Xm/Command.h>

Widget XmCommandGetChild(
    Widget *widget***,**
    unsigned char *child***);**

Description

XmCommandGetChild is used to access a component within a Command. The parameters given to the function are the Command widget and a value indicating which component to access.

widget Specifies the Command widget ID.

child Specifies a component within the Command. The following values are legal for this parameter:

- **XmDIALOG_COMMAND_TEXT**
- **XmDIALOG_PROMPT_LABEL**
- **XmDIALOG_HISTORY_LIST**
- **XmDIALOG_WORK_AREA**

For a complete definition of Command and its associated resources, see **XmCommand**(3).

Return Values

Returns the widget ID of the specified Command component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmCommand(3).

XmCommandSetValue

Purpose

A Command function that replaces a displayed string

Synopsis

#include <Xm/Command.h>

void XmCommandSetValue(
    Widget *widget***,**
    XmString *command***);**

Description

XmCommandSetValue replaces the string displayed in the command area of the Command widget with the passed **XmString**.

widget	Specifies the Command widget ID
command	Specifies the passed **XmString**

For a complete definition of Command and its associated resources, see **XmCommand**(3).

Related Information

XmCommand(3).

XmContainerCopy

Purpose

Container widget function to copy primary selection to the clipboard

Synopsis

#include <Xm/Container.h>

Boolean XmContainerCopy(
    Widget *container***,**
    Time *timestamp***);**

Description

XmContainerCopy copies the primary selected container items to the clipboard. This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmCOPY**.

container	Specifies the Container widget ID.
timestamp	Specifies the server time at which to modify the selection value.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

The function returns False in the following cases: if the primary selection is NULL, if the widget does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmContainer(3).

XmContainerCopyLink

Purpose

Container widget function to copy links to the clipboard

Synopsis

#include <Xm/Container.h>

Boolean XmContainerCopyLink(
    Widget *container***,**
    Time *timestamp***);**

Description

XmContainerCopyLink copies links to the primary selected items to the clipboard. This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmLINK**. The Container widget itself does not copy any links; **XmNconvertCallback** procedures are responsible for copying the link to the clipboard and for taking any related actions.

container	Specifies the Container widget ID.
timestamp	Specifies the server time at which to modify the selection value.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

The function returns False in the following cases: if the primary selection is NULL, if the widget does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmContainer(3).

XmContainerCut

Purpose

Container widget function to move items to the clipboard

Synopsis

#include <Xm/Container.h>

Boolean XmContainerCut(
Widget *container***,**
Time *timestamp***);**

Description

XmContainerCut cuts the primary selected items to the clipboard. This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmMOVE**. If the transfer is successful, this routine then calls the **XmNconvertCallback** procedures for the *CLIPBOARD* selection and the *DELETE* target.

container	Specifies the Container widget ID.
timestamp	Specifies the server time at which to modify the selection value.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

The function returns False in the following cases: if the primary selection is NULL, if the widget does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmContainer(3).

XmContainerGetItemChildren

Purpose

Container widget function to find all children of an item

Synopsis

#include <Xm/Container.h>

int XmContainerGetItemChildren(
  Widget *container***,**
  Widget *item***,**
  WidgetList * *item_children***);**

Description

XmContainerGetItemChildren allocates a WidgetList and stores within it the widget IDs of all widgets that have *item* specified as the value of their **XmNentryParent** resource. The application programmer is responsible for freeing the allocated WidgetList using XtFree. The number of widget IDs returned in *item_children* is returned by the function. If no widgets specify *item* as the value of their **XmNentryParent** resource, the function returns zero and *item_children* is left unchanged.

container	Specifies the Container widget ID.
item	Specifies a widgetID within *container*.
item_children	Returned array of Widgets.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

This function returns a count of all widgets that have *item* specified as the value of their **XmNentryParent** resource.

Related Information

XmContainer(3).

XmContainerPaste

Purpose

Container widget function to insert items from the clipboard

Synopsis

#include <Xm/Container.h>

Boolean XmContainerPaste(
    Widget *container***);**

Description

XmContainerPaste requests data transfer from the clipboard selection to the Container. This routine calls the widget's **XmNdestinationCallback** procedures with the *selection* member of the **XmDestinationCallbackStruct** set to *CLIPBOARD* and with the *operation* member set to **XmCOPY**. The Container widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the clipboard selection and for taking any related actions.

container Specifies the Container widget ID.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

The function returns False if no data transfer takes place. Otherwise, it returns True.

Related Information

XmContainer(3).

XmContainerPasteLink

Purpose

Container widget function to insert links from the clipboard

Synopsis

#include <Xm/Container.h>

Boolean XmContainerPasteLink(
    Widget *container***);**

Description

XmContainerPasteLink requests data transfer from the clipboard selection to the Container. This routine calls the widget's **XmNdestinationCallback** procedures with the *selection* member of the **XmDestinationCallbackStruct** set to *CLIPBOARD* and with the *operation* member set to **XmLINK**. The Container widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the clipboard selection and for taking any related actions.

container Specifies the Container widget ID.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

The function returns False if no data transfer takes place. Otherwise, it returns True.

Related Information

XmContainer(3).

XmContainerRelayout

Purpose

Container widget relayout function

Synopsis

#include <Xm/Container.h>

void XmContainerRelayout(
    Widget *container***);**

Description

XmContainerRelayout forces a layout of all items in the Container using the **XmNpositionIndex** and **XmNentryParent** constraint resources associated with each item.

container Specifies the Container widget ID.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Related Information

XmContainer(3).

XmContainerReorder

Purpose

Container widget function to reorder children

Synopsis

#include <Xm/Container.h>

void XmContainerReorder(
    Widget *container***,**
    WidgetList *widgets***,**
    int *num_widgets***);**

Description

XmContainerReorder obtains the **XmNpositionIndex** constraint resources of each widget specified in *widgets*, sorts them in ascending order, and inserts the **XmNpositionIndex** constraint resources in the new order into each widget. If the **XmNlayoutType** resource of Container is **XmOUTLINE** or **XmDETAIL**, **XmContainerReorder** will force a layout of all items.

container	Specifies the Container widget ID.
widgets	Specifies an array of widget children of *container*.
num_widgets	Specifies the number of items in the *widgets* array.

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Related Information

XmContainer(3).

XmConvertStringToUnits

Purpose

A function that converts a string specification to a unit value

Synopsis

#include <Xm/Xm.h>

int XmConvertStringToUnits(
    Screen ****screen*,**
    String *spec***,**
    int *orientation***,**
    int *to_type***,**
    XtEnum ****parse_error*);**

Description

XmConvertStringToUnits converts a string specification value and returns the converted value as the return value from the function. This function uses the specified screen's resolution to compute the number of units for the string specification.

screen	Specifies the screen whose resolution is to be used for the computation.
spec	Specifies the string, in <*floating value*><*unit*> format, to be converted.
orientation	Specifies whether the converter uses the horizontal or vertical screen resolution when performing the conversion. The *orientation* parameter can have values of **XmHORIZONTAL** or **XmVERTICAL**.
to_type	Converts the value to the unit type specified. Refer to the **XmNunitType** resource of the **XmGadget**, **XmManager**, or **XmPrimitive** reference page. This parameter can have one of the following values:

XmPIXELS	The returned value will be the number of pixels.
XmMILLIMETERS	The returned value will be the number of millimeters.

Xm100TH_MILLIMETERS
The returned values will be the number of 1/100 millimeters.

XmCENTIMETERS
The returned values will be the number of centimeters.

XmINCHES
The returned values will be the number of inches.

Xm1000TH_INCHES
The returned values will be the number of 1/100 inches.

XmPOINTS
The returned values will be the number of points. A point is a text processing unit defined as 1/72 of an inch.

Xm100TH_POINTS
The returned values will be the number of 1/100 points.

XmFONT_UNITS
All values provided to the widget are treated as font units. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

parse_error
Specifies if a parsing error occurred. This is set to a value of True indicates that an error occurred, a value of False to indicate no error.

Return Values

Returns the converted value. If a NULL screen, incorrect *orientation*, or incorrect *unit_type* is supplied as parameter data, or if a parsing error occurred, 0 (zero) is returned.

Related Information

XmConvertUnits(3), **XmSetFontUnits**(3), and **XmScreen**(3).

XmConvertUnits

Purpose

A function that converts a value in one unit type to another unit type

Synopsis

#include <Xm/Xm.h>

int XmConvertUnits(
  Widget *widget***,**
  int *orientation***,**
  int *from_unit_type***,**
  int *from_value***,**
  int *to_unit_type***);**

Description

XmConvertUnits converts the value and returns it as the return value from the function. For resources of type, dimension, or position, you can specify units using the syntax described in the **XmNunitType** resource of the **XmPrimitive** reference page.

widget Specifies the widget for which the data is to be converted.

orientation Specifies whether the converter uses the horizontal or vertical screen resolution when performing the conversions. The *orientation* parameter can have values of **XmHORIZONTAL** or **XmVERTICAL**.

from_unit_type
Specifies the current unit type of the supplied value

from_value Specifies the value to be converted

to_unit_type Converts the value to the unit type specified

The parameters *from_unit_type* and *to_unit_type* can have the following values:

XmPIXELS
All values provided to the widget are treated as pixel values. This is the default for the resource.

XmMILLIMETERS
All values provided to the widget are treated as millimeter values.

Xm100TH_MILLIMETERS
All values provided to the widget are treated as 1/100 of a millimeter.

XmCENTIMETERS
All values provided to the widget are treated as centimeter values.

XmINCHES
All values provided to the widget are treated as inch values.

Xm1000TH_INCHES
All values provided to the widget are treated as 1/1000 of an inch.

XmPOINTS
All values provided to the widget are treated as point values. A point is a unit used in text processing applications and is defined as 1/72 of an inch.

Xm100TH_POINTS
All values provided to the widget are treated as 1/100 of a point. A point is a unit typically used in text processing applications and is defined as 1/72 of an inch.

XmFONT_UNITS
All values provided to the widget are treated as normal font units. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

Return Values

Returns the converted value. If a NULL widget, incorrect *orientation*, or incorrect *unit_type* is supplied as parameter data, 0 (zero) is returned.

Related Information

XmPrimitive, **XmSetFontUnits**(3), and **XmScreen**(3).

XmCreateArrowButton

Purpose

The ArrowButton widget creation function

Synopsis

#include <Xm/ArrowB.h>

Widget XmCreateArrowButton(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateArrowButton creates an instance of an ArrowButton widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ArrowButton and its associated resources, see **XmArrowButton**(3).

Return Values

Returns the ArrowButton widget ID.

Related Information

XmArrowButton(3).

XmCreateArrowButtonGadget

Purpose

The ArrowButtonGadget creation function

Synopsis

#include <Xm/ArrowBG.h>

Widget XmCreateArrowButtonGadget(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateArrowButtonGadget creates an instance of an ArrowButtonGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ArrowButtonGadget and its associated resources, see **XmArrowButtonGadget**(3).

Return Values

Returns the ArrowButtonGadget widget ID.

Related Information

XmArrowButtonGadget(3).

XmCreateBulletinBoard

Purpose

The BulletinBoard widget creation function

Synopsis

#include <Xm/BulletinB.h>

Widget XmCreateBulletinBoard(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateBulletinBoard creates an instance of a BulletinBoard widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of BulletinBoard and its associated resources, see **XmBulletinBoard**(3).

Return Values

Returns the BulletinBoard widget ID.

Related Information

XmBulletinBoard(3).

XmCreateBulletinBoardDialog

Purpose

The BulletinBoard BulletinBoardDialog convenience creation function

Synopsis

#include <Xm/BulletinB.h>

Widget XmCreateBulletinBoardDialog(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateBulletinBoardDialog is a convenience creation function that creates a DialogShell and an unmanaged BulletinBoard child of the DialogShell. A BulletinBoardDialog is used for interactions not supported by the standard dialog set. This function does not automatically create any labels, buttons, or other dialog components. Such components should be added by the application after the BulletinBoardDialog is created.

Use **XtManageChild** to pop up the BulletinBoardDialog (passing the BulletinBoard as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateBulletinBoardDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of BulletinBoard and its associated resources, see **XmBulletinBoard**(3).

Return Values

Returns the BulletinBoard widget ID.

Related Information

XmBulletinBoard(3).

XmCreateCascadeButton

Purpose

The CascadeButton widget creation function

Synopsis

#include <Xm/CascadeB.h>

Widget XmCreateCascadeButton(
        Widget *parent***,**
        String *name***,**
        ArgList *arglist***,**
        Cardinal *argcount***);**

Description

XmCreateCascadeButton creates an instance of a CascadeButton widget and returns the associated widget ID.

parent	Specifies the parent widget ID. The parent must be a RowColumn widget.
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of CascadeButton and its associated resources, see **XmCascadeButton**(3).

Return Values

Returns the CascadeButton widget ID.

Related Information

XmCascadeButton(3).

XmCreateCascadeButtonGadget

Purpose

The CascadeButtonGadget creation function

Synopsis

#include <Xm/CascadeBG.h>

Widget XmCreateCascadeButtonGadget(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateCascadeButtonGadget creates an instance of a CascadeButtonGadget and returns the associated widget ID.

parent	Specifies the parent widget ID. The parent must be a RowColumn widget.
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of CascadeButtonGadget and its associated resources, see **XmCascadeButtonGadget**(3).

Return Values

Returns the CascadeButtonGadget widget ID.

Related Information

XmCascadeButtonGadget(3).

XmCreateComboBox

Purpose

The default ComboBox widget creation function

Synopsis

#include <Xm/ComboBox.h>

Widget XmCreateComboBox(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *arg_count***);**

Description

XmCreateComboBox creates an instance of a ComboBox widget of *XmNcomboBoxType XmCOMBO_BOX* and returns the associated widget ID.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
arg_count	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of ComboBox and its associated resources, see **XmComboBox**(3).

Return Values

Returns the ComboBox widget ID.

Related Information

XmComboBox(3).

XmCreateCommand

Purpose

The Command widget creation function

Synopsis

#include <Xm/Command.h>

Widget XmCreateCommand(
 Widget *parent***,**
 String *name***,**
 ArgList *arglist***,**
 Cardinal *argcount***);**

Description

XmCreateCommand creates an instance of a Command widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Command and its associated resources, see **XmCommand**(3).

Return Values

Returns the Command widget ID.

Related Information

XmCommand(3).

XmCreateContainer

Purpose

The Container widget creation function

Synopsis

#include <Xm/Container.h>

Widget XmCreateContainer(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateContainer creates an instance of a Container widget and returns the associated widget ID.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of Container and its associated resources, see **XmContainer**(3).

Return Values

This function returns the Container widget ID.

Related Information

XmContainer(3).

XmCreateDialogShell

Purpose

The DialogShell widget creation function

Synopsis

#include <Xm/DialogS.h>

```
Widget XmCreateDialogShell(
        Widget parent,
        String name,
        ArgList arglist,
        Cardinal argcount);
```

Description

XmCreateDialogShell creates an instance of a DialogShell widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DialogShell and its associated resources, see **XmDialogShell**(3).

Return Values

Returns the DialogShell widget ID.

Related Information

XmDialogShell(3).

XmCreateDragIcon

Purpose

A Drag and Drop function that creates a DragIcon widget

Synopsis

#include <Xm/DragIcon.h>

Widget XmCreateDragIcon(
    Widget *widget***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateDragIcon creates a DragIcon and returns the associated widget ID.

widget	Specifies the ID of the widget that the function uses to access default values for visual attributes of the DragIcon. This widget may be different than the actual parent of the DragIcon.
name	Specifies the name of the DragIcon widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of DragIcon and its associated resources, see **XmDragIcon**(3).

Return Values

The function creates a DragIcon and returns the associated widget ID.

Related Information

XmDragContext(3), **XmDragIcon**(3), and **XmScreen**(3).

XmCreateDrawingArea

Purpose

The DrawingArea widget creation function

Synopsis

#include <Xm/DrawingA.h>

Widget XmCreateDrawingArea(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateDrawingArea creates an instance of a DrawingArea widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DrawingArea and its associated resources, see **XmDrawingArea**(3).

Return Values

Returns the DrawingArea widget ID.

Related Information

XmDrawingArea(3).

XmCreateDrawnButton

Purpose

The DrawnButton widget creation function

Synopsis

#include <Xm/DrawnB.h>

Widget XmCreateDrawnButton(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateDrawnButton creates an instance of a DrawnButton widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DrawnButton and its associated resources, see **XmDrawnButton**(3).

Return Values

Returns the DrawnButton widget ID.

Related Information

XmDrawnButton(3).

XmCreateDropDownComboBox

Purpose

The Drop-down ComboBox widget creation function

Synopsis

#include <Xm/ComboBox.h>

Widget XmCreateDropDownComboBox(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *arg_count***);**

Description

XmCreateDropDownComboBox creates an instance of a ComboBox widget of *XmNcomboBoxType XmDROP_DOWN_COMBO_BOX* and returns the associated widget ID.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
arg_count	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of ComboBox and its associated resources, see **XmComboBox**(3).

Return Values

Returns the ComboBox widget ID.

Related Information

XmComboBox(3).

XmCreateDropDownList

Purpose

The Drop-down list ComboBox widget creation function

Synopsis

#include <Xm/ComboBox.h>

Widget XmCreateDropDownList(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *arg_count***);**

Description

XmCreateDropDownList creates an instance of a ComboBox widget of *XmNcomboBoxType XmDROP_DOWN_LIST* and returns the associated widget ID.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
arg_count	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of ComboBox and its associated resources, see **XmComboBox**(3).

Return Values

Returns the ComboBox widget ID.

Related Information

XmComboBox(3).

XmCreateErrorDialog

Purpose

The MessageBox ErrorDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateErrorDialog(
        Widget *parent***,**
        String *name***,**
        ArgList *arglist***,**
        Cardinal *argcount***);**

Description

XmCreateErrorDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. An ErrorDialog warns the user of an invalid or potentially dangerous condition. It includes a symbol, a message, and three buttons. The default symbol is an octagon with a diagonal slash. The default button labels are *OK*, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the ErrorDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateErrorDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateFileSelectionBox

Purpose

The FileSelectionBox widget creation function

Synopsis

#include <Xm/FileSB.h>

Widget XmCreateFileSelectionBox(
 Widget *parent***,**
 String *name***,**
 ArgList *arglist***,**
 Cardinal *argcount***);**

Description

XmCreateFileSelectionBox creates an unmanaged FileSelectionBox. A FileSelectionBox is used to select a file and includes the following:

- An editable text field for the directory mask
- A scrolling list of filenames
- An editable text field for the selected file
- Labels for the list and text fields
- Four buttons

The default button labels are *OK*, **Filter**, **Cancel**, and **Help**. Additional work area children may be added to the FileSelectionBox after creation. FileSelectionBox inherits the layout functionality provided by SelectionBox for any additional work area children.

If the parent of the FileSelectionBox is a DialogShell, use **XtManageChild** to pop up the FileSelectionDialog (passing the FileSelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox**(3).

Return Values

Returns the FileSelectionBox widget ID.

Related Information

XmFileSelectionBox(3).

XmCreateFileSelectionDialog

Purpose

The FileSelectionBox FileSelectionDialog convenience creation function

Synopsis

#include <Xm/FileSB.h>

Widget XmCreateFileSelectionDialog(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateFileSelectionDialog is a convenience creation function that creates a DialogShell and an unmanaged FileSelectionBox child of the DialogShell. A FileSelectionDialog selects a file. It includes the following:

- An editable text field for the directory mask
- A scrolling list of filenames
- An editable text field for the selected file
- Labels for the list and text fields
- Four buttons

The default button labels are *OK*, **Filter**, **Cancel**, and **Help**. One additional **WorkArea** child may be added to the FileSelectionBox after creation.

Use **XtManageChild** to pop up the FileSelectionDialog (passing the FileSelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateFileSelectionDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox**(3).

Return Values

Returns the FileSelectionBox widget ID.

Related Information

XmFileSelectionBox(3).

XmCreateForm

Purpose

The Form widget creation function

Synopsis

#include <Xm/Form.h>

Widget XmCreateForm(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateForm creates an instance of a Form widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Form and its associated resources, see **XmForm**(3).

Return Values

Returns the Form widget ID.

Related Information

XmForm(3).

XmCreateFormDialog

Purpose

A Form FormDialog convenience creation function

Synopsis

#include <Xm/Form.h>

Widget XmCreateFormDialog(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateFormDialog is a convenience creation function that creates a DialogShell and an unmanaged Form child of the DialogShell. A FormDialog is used for interactions not supported by the standard dialog set. This function does not automatically create any labels, buttons, or other dialog components. Such components should be added by the application after the FormDialog is created.

Use **XtManageChild** to pop up the FormDialog (passing the Form as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateFormDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Form and its associated resources, see **XmForm**(3).

Return Values

Returns the Form widget ID.

Related Information

XmForm(3).

XmCreateFrame

Purpose

The Frame widget creation function

Synopsis

#include <Xm/Frame.h>

Widget XmCreateFrame(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateFrame creates an instance of a Frame widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Frame and its associated resources, see **XmFrame**(3).

Return Values

Returns the Frame widget ID.

Related Information

XmFrame(3).

XmCreateIconGadget

Purpose

The IconGadget widget creation function

Synopsis

#include <Xm/IconG.h>

Widget XmCreateIconGadget(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateIconGadget creates an instance of an IconGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of IconGadget and its associated resources, see **XmIconGadget**(3).

Return Values

Returns the IconGadget widget ID.

Related Information

XmIconGadget(3).

XmCreateInformationDialog

Purpose

The MessageBox InformationDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateInformationDialog(
    Widget *parent*,
    String *name*,
    ArgList *arglist*,
    Cardinal *argcount*);

Description

XmCreateInformationDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. An InformationDialog gives the user information, such as the status of an action. It includes a symbol, a message, and three buttons. The default symbol is **i**. The default button labels are *OK*, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the InformationDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateInformationDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateLabel

Purpose

The Label widget creation function

Synopsis

#include <Xm/Label.h>

Widget XmCreateLabel(
　　Widget *parent***,**
　　String *name***,**
　　ArgList *arglist***,**
　　Cardinal *argcount***);**

Description

XmCreateLabel creates an instance of a Label widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Label and its associated resources, see **XmLabel**(3).

Return Values

Returns the Label widget ID.

Related Information

XmLabel(3).

XmCreateLabelGadget

Purpose

The LabelGadget creation function

Synopsis

#include <Xm/LabelG.h>

Widget XmCreateLabelGadget(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateLabelGadget creates an instance of a LabelGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of LabelGadget and its associated resources, see **XmLabelGadget**(3).

Return Values

Returns the LabelGadget widget ID.

Related Information

XmLabelGadget(3).

XmCreateList

Purpose

The List widget creation function

Synopsis

#include <Xm/List.h>

Widget XmCreateList(
  Widget *parent*,
  String *name*,
  ArgList *arglist*,
  Cardinal *argcount*);

Description

XmCreateList creates an instance of a List widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns the List widget ID.

Related Information

XmList(3).

XmCreateMainWindow

Purpose

The MainWindow widget creation function

Synopsis

#include <Xm/MainW.h>

Widget XmCreateMainWindow(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateMainWindow creates an instance of a MainWindow widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MainWindow and its associated resources, see **XmMainWindow**(3).

Return Values

Returns the MainWindow widget ID.

Related Information

XmMainWindow(3).

XmCreateMenuBar

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateMenuBar(
Widget *parent*,
String *name*,
ArgList *arglist*,
Cardinal *argcount*);

Description

XmCreateMenuBar creates an instance of a RowColumn widget of type **XmMENU_BAR** and returns the associated widget ID. It is provided as a convenience function for creating RowColumn widgets configured to operate as a MenuBar and is not implemented as a separate widget class.

The MenuBar widget is generally used for building a Pulldown menu system. Typically, a MenuBar is created and placed along the top of the application window, and several CascadeButtons are inserted as the children. Each of the CascadeButtons has a Pulldown menu pane associated with it. These Pulldown menu panes must have been created as children of the MenuBar. The user interacts with the MenuBar by using either the mouse or the keyboard.

The MenuBar displays a 3-D shadow along its border. The application controls the shadow attributes using the visual-related resources supported by **XmManager**.

The MenuBar widget is homogeneous in that it accepts only children that are a subclass of **XmCascadeButton** or **XmCascadeButtonGadget**. Attempting to insert a child of a different class results in a warning message.

If the MenuBar does not have enough room to fit all of its subwidgets on a single line, the MenuBar attempts to wrap the remaining entries onto additional lines if allowed by the geometry manager of the parent widget.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCascadeButton(3), **XmCascadeButtonGadget**(3), **XmCreatePulldownMenu**(3), **XmCreateSimpleMenuBar**(3), **XmManager**(3), **XmRowColumn**(3), and **XmVaCreateSimpleMenuBar**(3).

XmCreateMenuShell

Purpose

The MenuShell widget creation function

Synopsis

#include <Xm/MenuShell.h>

Widget XmCreateMenuShell(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateMenuShell creates an instance of a MenuShell widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MenuShell and its associated resources, see **XmMenuShell**(3).

Return Values

Returns the MenuShell widget ID.

Related Information

XmMenuShell(3).

XmCreateMessageBox

Purpose

The MessageBox widget creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateMessageBox(
　　Widget *parent***,**
　　String *name***,**
　　ArgList *arglist***,**
　　Cardinal *argcount***);**

Description

XmCreateMessageBox creates an unmanaged MessageBox. A MessageBox is used for common interaction tasks, which include giving information, asking questions, and reporting errors. It includes an optional symbol, a message, and three buttons.

By default, there is no symbol. The default button labels are *OK*, **Cancel**, and **Help**.

If the parent of the MessageBox is a DialogShell, use **XtManageChild** to pop up the MessageBox (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateMessageDialog

Purpose

The MessageBox MessageDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateMessageDialog(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateMessageDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A MessageDialog is used for common interaction tasks, which include giving information, asking questions, and reporting errors. It includes a symbol, a message, and three buttons. By default, there is no symbol. The default button labels are *OK*, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the MessageDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateMessageDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateNotebook

Purpose

The Notebook widget creation function

Synopsis

#include <Xm/Notebook.h>

void XmCreateNotebook(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateNotebook creates an instance of a Notebook widget and returns the associated widget ID.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of Notebook and its associated resources, see **XmNotebook**(3).

Return Values

Returns the Notebook widget ID.

Related Information

XmNotebook(3).

XmCreateOptionMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateOptionMenu(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateOptionMenu creates an instance of a RowColumn widget of type **XmMENU_OPTION** and returns the associated widget ID.

It is provided as a convenience function for creating a RowColumn widget configured to operate as an OptionMenu and is not implemented as a separate widget class.

The OptionMenu widget is a specialized RowColumn manager composed of a label, a selection area, and a single Pulldown menu pane. When an application creates an OptionMenu widget, it supplies the label string and the Pulldown menu pane. In order for the operation to be successful, there must be a valid **XmNsubMenuId** resource set when this function is called. The LabelGadget and the selection area (a CascadeButtonGadget) are created by the OptionMenu.

The OptionMenu's Pulldown menu pane must not contain any ToggleButtons or ToggleButtonGadgets. The results of including CascadeButtons or CascadeButtonGadgets in the OptionMenu's Pulldown menu pane are undefined.

An OptionMenu is laid out with the label displayed on one side of the widget and the selection area on the other side when **XmNorientation** is *XmHORIZONTAL*. The layout of the label with respect to the selection area depends on the **XmNlayoutDirection** resource in the horizontal orientation. If the value is **XmVERTICAL**, the label is above the selection area. The selection area has a dual purpose; it displays the label of the last item selected from the associated Pulldown menu pane, and it provides the means for posting the Pulldown menu pane.

The OptionMenu typically does not display any 3-D visuals around itself or the internal LabelGadget. By default, the internal CascadeButtonGadget has a visible 3-D shadow. The application may change this by getting the CascadeButtonGadget ID using **XmOptionButtonGadget**, and then calling **XtSetValues** using the standard visual-related resources.

The Pulldown menu pane is posted when the mouse pointer is moved over the selection area and a mouse button that is defined by OptionMenu's RowColumn parent is pressed. The Pulldown menu pane is posted and positioned so that the last selected item is directly over the selection area. The mouse is then used to arm the desired menu item. When the mouse button is released, the armed menu item is selected and the label within the selection area is changed to match that of the selected item. By default, **BSelect** is used to interact with an OptionMenu. The default can be changed with the RowColumn resource **XmNmenuPost**.

The OptionMenu also operates with the keyboard interface mechanism. If the application has established a mnemonic with the OptionMenu, pressing **Alt** with the mnemonic causes the Pulldown menu pane to be posted with traversal enabled. The standard traversal keys can then be used to move within the menu pane. Pressing **Return** or typing a mnemonic or accelerator for one of the menu items selects that item.

An application may use the **XmNmenuHistory** resource to indicate which item in the Pulldown menu pane should be treated as the current choice and have its label displayed in the selection area. By default, the first selectable item in the Pulldown menu pane is used.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. These widgets (or gadgets) and the associated OptionMenu areas are

Option Menu Label Gadget
OptionLabel

Option Menu Cascade Button
OptionButton

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCascadeButtonGadget(3), **XmCreatePulldownMenu**(3), **XmCreateSimpleOptionMenu**(3), **XmLabelGadget**(3), **XmOptionButtonGadget**(3), **XmOptionLabelGadget**(3), **XmRowColumn**(3), and **XmVaCreateSimpleOptionMenu**(3).

XmCreatePanedWindow

Purpose

The PanedWindow widget creation function

Synopsis

#include <Xm/PanedW.h>

Widget XmCreatePanedWindow(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreatePanedWindow creates an instance of a PanedWindow widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of PanedWindow and its associated resources, see **XmPanedWindow**(3).

Return Values

Returns the PanedWindow widget ID.

Related Information

XmPanedWindow(3).

XmCreatePopupMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreatePopupMenu(
    Widget *parent*,
    String *name*,
    ArgList *arglist*,
    Cardinal *argcount*);

Description

XmCreatePopupMenu creates an instance of a RowColumn widget of type **XmMENU_POPUP** and returns the associated widget ID. When this function is used to create the Popup menu pane, a MenuShell widget is automatically created as the parent of the menu pane. The parent of the MenuShell widget is the widget indicated by the *parent* parameter.

XmCreatePopupMenu is provided as a convenience function for creating RowColumn widgets configured to operate as Popup menu panes and is not implemented as a separate widget class.

The PopupMenu is used as the first menu pane within a PopupMenu system; all other menu panes are of the Pulldown type. A Popup menu pane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the menu pane.

The Popup menu pane must be created as the child of a MenuShell widget in order to function properly when it is incorporated into a menu. If the application uses this convenience function for creating a Popup menu pane, the MenuShell is automatically created as the real parent of the menu pane. If the application does not use this convenience function to create the RowColumn to function as a Popup menu pane, it is the application's responsibility to create the MenuShell widget.

To access the PopupMenu, the application must first position the widget using the **XmMenuPosition** function and then manage it using **XtManageChild**.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

Popup menu panes support tear-off capabilities for tear-off menus through **XmRowColumn** resources. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateSimplePopupMenu(3), **XmMenuPosition**(3), **XmMenuShell**(3), **XmRowColumn**(3), and **XmVaCreateSimplePopupMenu**(3).

XmCreatePromptDialog

Purpose

The SelectionBox PromptDialog convenience creation function

Synopsis

#include <Xm/SelectioB.h>

Widget XmCreatePromptDialog(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreatePromptDialog is a convenience creation function that creates a DialogShell and an unmanaged SelectionBox child of the DialogShell. A PromptDialog prompts the user for text input. It includes a message, a text input region, and three managed buttons. The default button labels are *OK*, **Cancel**, and **Help**. An additional button, with **Apply** as the default label, is created unmanaged; it may be explicitly managed if needed. One additional **WorkArea** child may be added to the SelectionBox after creation.

XmCreatePromptDialog forces the value of the SelectionBox resource **XmNdialogType** to **XmDIALOG_PROMPT**.

Use **XtManageChild** to pop up the PromptDialog (passing the SelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreatePromptDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox**(3).

Return Values

Returns the SelectionBox widget ID.

Related Information

XmSelectionBox(3).

XmCreatePulldownMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreatePulldownMenu(
    Widget *parent*,
    String *name*,
    ArgList *arglist*,
    Cardinal *argcount*);

Description

XmCreatePulldownMenu creates an instance of a RowColumn widget of type **XmMENU_PULLDOWN** and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

Specifies the number of attribute/value pairs in the argument list (*arglist*). When this function is used to create the Pulldown menu pane, a MenuShell widget is automatically created as the parent of the menu pane. If the widget specified by the *parent* parameter is a Popup or a Pulldown menu pane, the MenuShell widget is created as a child of the *parent* MenuShell; otherwise, it is created as a child of the specified *parent* widget.

XmCreatePulldownMenu is provided as a convenience function for creating RowColumn widgets configured to operate as Pulldown menu panes and is not implemented as a separate widget class.

A Pulldown menu pane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the menu pane.

A Pulldown menu pane is used with submenus that are to be attached to a CascadeButton or a CascadeButtonGadget. This is the case for all menu panes that are part of a PulldownMenu system (a MenuBar), the menu pane associated with an OptionMenu, and any menu panes that cascade from a Popup menu pane. Pulldown menu panes that are to be associated with an OptionMenu must be created before the OptionMenu is created.

The Pulldown menu pane must be attached to a CascadeButton or CascadeButtonGadget that resides in a MenuBar, a Popup menu pane, a Pulldown menu pane, or an OptionMenu. It is attached with the button resource **XmNsubMenuId**.

A MenuShell widget is required between the Pulldown menu pane and its parent. If the application uses this convenience function for creating a Pulldown menu pane, the MenuShell is automatically created as the real parent of the menu pane; otherwise, it is the application's responsibility to create the MenuShell widget.

To function correctly when incorporated into a menu, the Pulldown menu pane's hierarchy must be considered. This hierarchy depends on the type of menu system that is being built, as follows:

- If the Pulldown menu pane is to be pulled down from a MenuBar, its *parent* must be the MenuBar.
- If the Pulldown menu pane is to be pulled down from a Popup or another Pulldown menu pane, its *parent* must be that Popup or Pulldown menu pane.
- If the Pulldown menu pane is to be pulled down from an OptionMenu, its *parent* must be the same as the OptionMenu parent.

PullDown menu panes support tear-off capabilities for tear-off menus through **XmRowColumn** resources. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCascadeButton(3), **XmCascadeButtonGadget**(3), **XmCreateOptionMenu**(3), **XmCreatePopupMenu**(3), **XmCreateSimplePulldownMenu**(3), **XmMenuShell**(3), **XmRowColumn**(3), and **XmVaCreateSimplePulldownMenu**(3).

XmCreatePushButton

Purpose

The PushButton widget creation function

Synopsis

#include <Xm/PushB.h>

Widget XmCreatePushButton(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreatePushButton creates an instance of a PushButton widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of PushButton and its associated resources, see **XmPushButton**(3).

Return Values

Returns the PushButton widget ID.

Related Information

XmPushButton(3).

XmCreatePushButtonGadget

Purpose

The PushButtonGadget creation function

Synopsis

#include <Xm/PushBG.h>

Widget XmCreatePushButtonGadget(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreatePushButtonGadget creates an instance of a PushButtonGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of PushButtonGadget and its associated resources, see **XmPushButtonGadget**(3).

Return Values

Returns the PushButtonGadget widget ID.

Related Information

XmPushButtonGadget(3).

XmCreateQuestionDialog

Purpose

The MessageBox QuestionDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateQuestionDialog(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateQuestionDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A QuestionDialog is used to get the answer to a question from the user. It includes a symbol, a message, and three buttons. The default symbol is a question mark. The default button labels are *OK*, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the QuestionDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateQuestionDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateRadioBox

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateRadioBox(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateRadioBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID. Typically, this is a composite widget that contains multiple ToggleButtonGadgets. The RadioBox arbitrates and ensures that at most one ToggleButtonGadget is on at any time.

Unless the application supplies other values in the *arglist*, this function provides initial values for several RowColumn resources. It initializes **XmNpacking** to **XmPACK_COLUMN**, **XmNradioBehavior** to True, **XmNisHomogeneous** to True, and **XmNentryClass** to **XmToggleButtonGadgetClass**.

In a RadioBox, the ToggleButton or ToggleButtonGadget resource **XmNindicatorType** defaults to **XmONE_OF_MANY**, and the ToggleButton or ToggleButtonGadget resource**XmNvisibleWhenOff** defaults to True.

This routine is provided as a convenience function for creating RowColumn widgets.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateRowColumn(3), **XmCreateSimpleCheckBox**(3), **XmCreateSimpleRadioBox**(3), **XmCreateWorkArea**(3), **XmRowColumn**(3), **XmVaCreateSimpleCheckBox**(3), and **XmVaCreateSimpleRadioBox**(3).

XmCreateRowColumn

Purpose

The RowColumn widget creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateRowColumn(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateRowColumn creates an instance of a RowColumn widget and returns the associated widget ID. If **XmNrowColumnType** is not specified, then it is created with **XmWORK_AREA**, which is the default.

If this function is used to create a Popup Menu of type **XmMENU_POPUP** or a Pulldown Menu of type **XmMENU_PULLDOWN**, a MenuShell widget is not automatically created as the parent of the menu pane. The application must first create the MenuShell by using either **XmCreateMenuShell** or the standard toolkit create function.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateMenuBar(3), **XmCreateMenuShell**(3), **XmCreateOptionMenu**(3), **XmCreatePopupMenu**(3), **XmCreatePulldownMenu**(3), **XmCreateRadioBox**(3), **XmCreateSimpleCheckBox**(3), **XmCreateSimpleMenuBar**(3), **XmCreateSimpleOptionMenu**(3), **XmCreateSimplePopupMenu**(3), **XmCreateSimplePulldownMenu**(3), **XmCreateSimpleRadioBox**(3), **XmCreateWorkArea**(3), **XmRowColumn**(3), **XmVaCreateSimpleCheckBox**(3), **XmVaCreateSimpleMenuBar**(3), **XmVaCreateSimpleOptionMenu**(3), **XmVaCreateSimplePopupMenu**(3), **XmVaCreateSimplePulldownMenu**(3), and **XmVaCreateSimpleRadioBox**(3).

XmCreateScale

Purpose

The Scale widget creation function

Synopsis

#include <Xm/Scale.h>

```
Widget XmCreateScale(
          Widget parent,
          String name,
          ArgList arglist,
          Cardinal argcount);
```

Description

XmCreateScale creates an instance of a Scale widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Scale and its associated resources, see **XmScale**(3).

Return Values

Returns the Scale widget ID.

Related Information

XmScale(3).

XmCreateScrollBar

Purpose

The ScrollBar widget creation function

Synopsis

#include <Xm/ScrollBar.h>

Widget XmCreateScrollBar(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateScrollBar creates an instance of a ScrollBar widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ScrollBar and its associated resources, see **XmScrollBar**(3).

Return Values

Returns the ScrollBar widget ID.

Related Information

XmScrollBar(3).

XmCreateScrolledList

Purpose

The List ScrolledList convenience creation function

Synopsis

#include <Xm/List.h>

Widget XmCreateScrolledList(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateScrolledList creates an instance of a List widget that is contained within a ScrolledWindow. The ScrolledWindow parent is created managed. All ScrolledWindow subarea widgets are automatically created by this function. The ID returned by this function is that of the List widget (not the ScrolledWindow widget). Use this widget ID for all operations on the List widget. Use the widget ID of the List widget's parent for all operations on the ScrolledWindow. To obtain the ID of the ScrolledWindow widget associated with the List widget, use the Xt Intrinsics **XtParent** function. The name of the ScrolledWindow created by this function is formed by concatenating *SW* onto the end of the *name* specified in the parameter list.

All arguments to either the List or the ScrolledWindow widget can be specified at creation time using this function. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the List or the ScrolledWindow widget as appropriate. Note that the result of providing the **XmNdestroyCallback** resource in the creation *arglist* is unspecified. The application should use the **XtAddCallback** function to add callbacks to the appropriate widget (List or ScrolledWindow) after creating it.

This function forces the following initial values for ScrolledWindow resources:

- **XmNscrollingPolicy** is set to **XmAPPLICATION_DEFINED**.
- **XmNvisualPolicy** is set to **XmVARIABLE**.

- **XmNscrollBarDisplayPolicy** is set to **XmSTATIC**. (No initial value is forced for the List's **XmNscrollBarDisplayPolicy**.)
- **XmNshadowThickness** is set to 0 (zero).

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns the List widget ID.

Related Information

XmList(3) and **XmScrolledWindow**(3).

XmCreateScrolledText

Purpose

The Text ScrolledText convenience creation function

Synopsis

#include <Xm/Text.h>

Widget XmCreateScrolledText(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateScrolledText creates an instance of a Text widget that is contained within a ScrolledWindow. The ScrolledWindow parent is created managed. All ScrolledWindow subarea widgets are automatically created by this function. The ID returned by this function is that of the Text widget (not the ScrolledWindow widget). Use this widget ID for all operations on the Text widget. Use the widget ID of the Text widget's parent for all operations on the ScrolledWindow. To obtain the ID of the ScrolledWindow widget associated with the Text widget, use the Xt Intrinsics **XtParent** function. The name of the ScrolledWindow created by this function is formed by concatenating the letters *SW* onto the end of the *name* specified in the parameter list.

The Text widget defaults to single-line text edit; therefore, no ScrollBars are displayed. The Text resource **XmNeditMode** must be set to **XmMULTI_LINE_EDIT** to display the ScrollBars. The results of placing a Text widget inside a ScrolledWindow when the Text's **XmNeditMode** is **XmSINGLE_LINE_EDIT** are undefined.

All arguments to either the Text or the ScrolledWindow widget can be specified at creation time with this function. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the Text or the ScrolledWindow widget as appropriate. Note that the result of providing the **XmNdestroyCallback** resource in the creation *arglist* is unspecified. The application should use the **XtAddCallback** function to add callbacks to the appropriate widget (Text or ScrolledWindow) after creating it.

This function forces the following initial values for ScrolledWindow resources:

- **XmNscrollingPolicy** is set to **XmAPPLICATION_DEFINED**.
- **XmNvisualPolicy** is set to **XmVARIABLE**.
- **XmNscrollBarDisplayPolicy** is set to **XmSTATIC**.
- **XmNshadowThickness** is set to 0 (zero).

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns the Text widget ID.

Related Information

XmScrolledWindow(3) and **XmText**(3).

XmCreateScrolledWindow

Purpose

The ScrolledWindow widget creation function

Synopsis

#include <Xm/ScrolledW.h>

Widget XmCreateScrolledWindow(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateScrolledWindow creates an instance of a ScrolledWindow widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ScrolledWindow and its associated resources, see **XmScrolledWindow**(3).

Return Values

Returns the ScrolledWindow widget ID.

Related Information

XmScrolledWindow(3).

XmCreateSelectionBox

Purpose

The SelectionBox widget creation function

Synopsis

#include <Xm/SelectioB.h>

Widget XmCreateSelectionBox(
    Widget *parent*,
    String *name*,
    ArgList *arglist*,
    Cardinal *argcount*);

Description

XmCreateSelectionBox creates an unmanaged SelectionBox. A SelectionBox is used to get a selection from a list of alternatives from the user and includes the following:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the list and text field
- Three or four buttons

The default button labels are *OK*, **Cancel**, and **Help**. By default, an **Apply** button is also created. If the parent of the SelectionBox is a DialogShell, it is managed; otherwise it is unmanaged. Additional work area children may be added to the SelectionBox after creation.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox**(3).

Return Values

Returns the SelectionBox widget ID.

Related Information

XmSelectionBox(3).

XmCreateSelectionDialog

Purpose

The SelectionBox SelectionDialog convenience creation function

Synopsis

#include <Xm/SelectioB.h>

Widget XmCreateSelectionDialog(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateSelectionDialog is a convenience creation function that creates a DialogShell and an unmanaged SelectionBox child of the DialogShell. A SelectionDialog offers the user a choice from a list of alternatives and gets a selection. It includes the following:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the text field
- Four buttons

The default button labels are *OK*, **Cancel**, **Apply**, and **Help**. One additional **WorkArea** child may be added to the SelectionBox after creation.

XmCreateSelectionDialog forces the value of the SelectionBox resource **XmNdialogType** to **XmDIALOG_SELECTION**.

XmCreateSelectionDialog forces the value of the Shell resource **XmNallowShellResize** to True.

Use **XtManageChild** to pop up the SelectionDialog (passing the SelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox**(3).

Return Values

Returns the SelectionBox widget ID.

Related Information

XmSelectionBox(3).

XmCreateSeparator

Purpose

The Separator widget creation function

Synopsis

#include <Xm/Separator.h>

Widget XmCreateSeparator(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateSeparator creates an instance of a Separator widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Separator and its associated resources, see **XmSeparator**(3).

Return Values

Returns the Separator widget ID.

Related Information

XmSeparator(3).

XmCreateSeparatorGadget

Purpose

The SeparatorGadget creation function

Synopsis

#include <Xm/SeparatoG.h>

Widget XmCreateSeparatorGadget(
 Widget *parent***,**
 String *name***,**
 ArgList *arglist***,**
 Cardinal *argcount***);**

Description

XmCreateSeparatorGadget creates an instance of a SeparatorGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SeparatorGadget and its associated resources, see **XmSeparatorGadget**(3).

Return Values

Returns the SeparatorGadget widget ID.

Related Information

XmSeparatorGadget(3).

XmCreateSimpleCheckBox

Purpose

A RowColumn widget convenience creation function

Synopsis

```
#include <Xm/RowColumn.h>

Widget XmCreateSimpleCheckBox(
        Widget parent,
        String name,
        ArgList arglist,
        Cardinal argcount);
```

Description

XmCreateSimpleCheckBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID.

This routine creates a CheckBox and its ToggleButtonGadget children. A CheckBox is similar to a RadioBox, except that more than one button can be selected at a time. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. Buttons are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button type allowed in the **XmNbuttonType** resource is **XmCHECKBUTTON**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3), **XmCreateRowColumn**(3), **XmCreateSimpleRadioBox**(3), **XmRowColumn**(3), **XmVaCreateSimpleCheckBox**(3), and **XmVaCreateSimpleRadioBox**(3).

XmCreateSimpleMenuBar

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateSimpleMenuBar(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateSimpleMenuBar creates an instance of a RowColumn widget of type **XmMENU_BAR** and returns the associated widget ID.

This routine creates a MenuBar and its CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. Buttons are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button type allowed in the **XmNbuttonType** resource is **XmCASCADEBUTTON**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateMenuBar(3), **XmCreateRowColumn**(3), **XmRowColumn**(3), and **XmVaCreateSimpleMenuBar**(3).

XmCreateSimpleOptionMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateSimpleOptionMenu(
Widget *parent***,**
String *name***,**
ArgList *arglist***,**
Cardinal *argcount***);**

Description

XmCreateSimpleOptionMenu creates an instance of a RowColumn widget of type **XmMENU_OPTION** and returns the associated widget ID.

This routine creates an OptionMenu and its submenu containing PushButtonGadget or CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to the number of separators in the menu minus 1. Buttons and separators are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. These widgets (or gadgets) and the associated OptionMenu areas are

Option Menu Label Gadget
OptionLabel

Option Menu Cascade Button
OptionButton

A number of resources exist specifically for use with this and other simple menu creation routines. The only button types allowed in the **XmNbuttonType** resource are **XmPUSHBUTTON**, **XmCASCADEBUTTON**, **XmSEPARATOR**, and **XmDOUBLE_SEPARATOR**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateOptionMenu(3), **XmCreateRowColumn**(3), **XmRowColumn**(3), and **XmVaCreateSimpleOptionMenu**(3).

XmCreateSimplePopupMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateSimplePopupMenu(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateSimplePopupMenu creates an instance of a RowColumn widget of type **XmMENU_POPUP** and returns the associated widget ID.

This routine creates a Popup menu pane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to the number of separators in the menu minus 1. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to the number of titles in the menu minus 1. Buttons, separators, and titles are named and created in the order in which they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the widget ID of the parent of the MenuShell
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button types allowed in the **XmNbuttonType** resource are **XmCASCADEBUTTON**, **XmPUSHBUTTON**, **XmRADIOBUTTON**, **XmCHECKBUTTON**, **XmTITLE**, **XmSEPARATOR**, and

XmDOUBLE_SEPARATOR. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreatePopupMenu(3), **XmCreateRowColumn**(3), **XmRowColumn**(3), and **XmVaCreateSimplePopupMenu**(3).

XmCreateSimplePulldownMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateSimplePulldownMenu(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateSimplePulldownMenu creates an instance of a RowColumn widget of type **XmMENU_PULLDOWN** and returns the associated widget ID.

This routine creates a Pulldown menu pane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to the number of separators in the menu minus 1. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to the number of titles in the menu minus 1. Buttons, separators, and titles are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the widget ID of the parent of the MenuShell
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button types allowed in the **XmNbuttonType** resource are **XmCASCADEBUTTON**, **XmPUSHBUTTON**, **XmRADIOBUTTON**, **XmCHECKBUTTON**, **XmTITLE**, **XmSEPARATOR**, and

XmDOUBLE_SEPARATOR. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreatePulldownMenu(3), **XmCreateRowColumn**(3), **XmRowColumn**(3), and **XmVaCreateSimplePulldownMenu**(3).

XmCreateSimpleRadioBox

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateSimpleRadioBox(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateSimpleRadioBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID.

This routine creates a RadioBox and its ToggleButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. Buttons are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button type allowed in the **XmNbuttonType** resource is **XmRADIOBUTTON**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3), **XmCreateRowColumn**(3),
XmCreateSimpleCheckBox(3), **XmRowColumn**(3), and
XmVaCreateSimpleRadioBox(3).

XmCreateSimpleSpinBox

Purpose

the SimpleSpinBox widget creation function

Synopsis

#include <Xm/SSpinB.h>

Widget XmCreateSimpleSpinBox(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

The **XmCreateSimpleSpinBox** function creates an instance of a SpinBox widget and returns the associated widget ID.

The *parent* argument specifies the parent widget ID.

The *name* argument specifies the name of the created widget.

The *arglist* argument specifies the argument list.

The *argcount* argument specifies the number of attribute/value pairs in the argument list.

Return Values

Upon successful completion, the **XmCreateSimpleSpinBox** function returns the SimpleSpinBox widget ID.

Related Information

XmSimpleSpinBox(3).

XmCreateSpinBox

Purpose

The SpinBox creation function

Synopsis

#include <Xm/SpinB.h>

Widget XmCreateSpinBox(
  Widget *parent***,**
  String *name***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmCreateSpinBox creates a SpinBox widget.

This function creates a SpinBox with two arrows, but without any traversable children (choices to spin). The application can create text children to go with this parent SpinBox using **XmCreateTextField** or **XmCreateText**.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SpinBox and its associated resources, see **XmSpinBox**(3).

Return Values

Returns the SpinBox widget ID.

Related Information

XmSpinBox(3)

XmCreateTemplateDialog

Purpose

A MessageBox TemplateDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateTemplateDialog(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateTemplateDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. The MessageBox widget's **XmNdialogType** resource is set to **XmDIALOG_TEMPLATE**. By default, the TemplateDialog widget contains only the separator child. You can build a customized dialog by adding children to the TemplateDialog.

You can create the standard MessageBox pushbuttons, **Cancel**, **Help**, and *OK*, by specifying the associated callback and label string resources. Setting **XmNsymbolPixmap** or **XmNmessageString** creates a symbol or message label.

Use **XtManageChild** to pop up the TemplateDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateTemplateDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateText

Purpose

The Text widget creation function

Synopsis

#include <Xm/Text.h>

Widget XmCreateText(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateText creates an instance of a Text widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns the Text widget ID.

Related Information

XmText(3).

XmCreateTextField

Purpose

The TextField widget creation function

Synopsis

#include <Xm/TextF.h>

Widget XmCreateTextField(
  Widget *parent*,
  String *name*,
  ArgList *arglist*,
  Cardinal *argcount*);

Description

XmCreateTextField creates an instance of a TextField widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns the TextField widget ID.

Related Information

XmTextField(3).

XmCreateToggleButton

Purpose

The ToggleButton widget creation function

Synopsis

#include <Xm/ToggleB.h>

Widget XmCreateToggleButton(
 Widget *parent***,**
 String *name***,**
 ArgList *arglist***,**
 Cardinal *argcount***);**

Description

XmCreateToggleButton creates an instance of a ToggleButton widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton**(3).

Return Values

Returns the ToggleButton widget ID.

Related Information

XmToggleButton(3).

XmCreateToggleButtonGadget

Purpose

The ToggleButtonGadget creation function

Synopsis

#include <Xm/ToggleBG.h>

Widget XmCreateToggleButtonGadget(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateToggleButtonGadget creates an instance of a ToggleButtonGadget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ToggleButtonGadget and its associated resources, see **XmToggleButtonGadget**(3).

Return Values

Returns the ToggleButtonGadget widget ID.

Related Information

XmToggleButtonGadget(3).

XmCreateWarningDialog

Purpose

The MessageBox WarningDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateWarningDialog(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateWarningDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A WarningDialog warns users of action consequences and gives them a choice of resolutions. It includes a symbol, a message, and three buttons. The default symbol is an exclamation point. The default button labels are *OK*, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the WarningDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateWarningDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCreateWorkArea

Purpose

A function that creates a RowColumn WorkArea

Synopsis

#include <Xm/RowColumn.h>

Widget XmCreateWorkArea(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateWorkArea creates an instance of a RowColumn widget and returns the associated widget ID. The widget is created with **XmNrowColumnType** set to **XmWORK_AREA**.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3), **XmCreateSimpleCheckBox**(3), **XmCreateSimpleRadioBox**(3), **XmRowColumn**(3), **XmVaCreateSimpleCheckBox**(3), and **XmVaCreateSimpleRadioBox**(3).

XmCreateWorkingDialog

Purpose

The MessageBox WorkingDialog convenience creation function

Synopsis

#include <Xm/MessageB.h>

Widget XmCreateWorkingDialog(
    Widget *parent***,**
    String *name***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmCreateWorkingDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A WorkingDialog informs users that there is a time-consuming operation in progress and allows them to cancel the operation. It includes a symbol, a message, and three buttons. The default symbol is an hourglass. The default button labels are *OK*, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the WorkingDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

XmCreateWorkingDialog forces the value of the Shell resource **XmNallowShellResize** to True.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3).

XmCvtByteStreamToXmString

Purpose

A compound string function that converts from a compound string in Byte Stream format to a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmCvtByteStreamToXmString(
        unsigned char ****property*);**

Description

XmCvtByteStreamToXmString converts a stream of bytes representing a compound string in Byte Stream format to a compound string. This routine is typically used by the destination of a data transfer operation to produce a compound string from a transferred Byte Stream representation.

property   Specifies a compound string representation in Byte Stream format.

Return Values

Returns a compound string. The function allocates space to hold the returned compound string. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XmStringFree**.

Related Information

XmString(3), **XmCvtXmStringToByteStream**(3), and **XmStringFree**(3).

XmCvtCTToXmString

Purpose

A compound string function that converts compound text to a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmCvtCTToXmString(
        char * *text***);**

Description

XmCvtCTToXmString converts a (**char ***) string in compound text format to a compound string. The application must call **XtAppInitialize** before calling this function. Conversion of compound text to compound strings is implementation dependent.

text	Specifies a string in compound text format to be converted to a compound string.

Return Values

Returns a compound string derived from the compound text. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**. The compound text is assumed to be NULL-terminated; NULLs within the compound text are handled correctly. The handling of HORIZONTAL TABULATION (HT) control characters within the compound text is undefined. The compound text format is described in the X Consortium Standard *Compound Text Encoding*.

Related Information

XmCvtXmStringToCT(3).

XmCvtStringToUnitType

Purpose

A function that converts a string to a unit-type value

Synopsis

#include <Xm/Xm.h>

void XmCvtStringToUnitType(
XrmValuePtr *args***,**
Cardinal * *num_args***,**
XrmValue * *from_val***,**
XrmValue * *to_val***);**

Description

XmCvtStringToUnitType converts a string to a unit type. Refer to the reference pages for **XmGadget**, **XmManager**, or **XmPrimitive** for a description of the valid unit types. Use of this function as a resource converter is obsolete. It has been replaced by a new resource converter that uses the RepType facility.

args	Specifies a list of additional *XrmValue* arguments to the converter if additional context is needed to perform the conversion. For example, the string-to-font converter needs the widget's screen and the string-to-pixel converter needs the widget's screen and color map. This argument is often NULL.
num_args	Specifies the number of additional *XrmValue* arguments. This argument is often zero.
from_val	Specifies the value to convert
to_val	Specifies the descriptor to use to return the converted value

Related Information

XmGadget(3), **XmManager**(3), and **XmPrimitive**(3).

XmCvtTextPropertyToXmStringTable

Purpose

A function that converts from a TextProperty Structure to a StringTable

Synopsis

#include <Xm/Xm.h>
int XmCvtTextPropertyToXmStringTable (*display, text_prop, string_table_return, count_return***)**
Display ****display*;**
XTextProperty ****text_prop*;**
XmStringTable ****string_table_return*;**
int ****count_return*;**

Description

XmCvtTextPropertyToXmStringTable converts the specified *XTextProperty* structure into an **XmStringTable**, as follows:

- If the encoding member of *text_prop* is the Atom *STRING*, each returned **XmString** has a tag of "ISO8859-1" and a text type of **XmCHARSET_TEXT**.
- If the encoding member of *text_prop* is the encoding of the current locale, and if that encoding is not *STRING*, each returned **XmString** has a tag of **_MOTIF_DEFAULT_LOCALE** and a text type of **XmMULTIBYTE_TEXT**.
- If the encoding member of *text_prop* is other than *STRING* or the encoding of the current locale, the contents of the returned compound strings are implementation dependent.

If conversion depends on the locale and the current locale is not supported, the function returns **XLocaleNotSupported**. If conversion to the encoding of the current locale is required and if the locale is supported but no converter is available for the encoding specified in *text_prop*, the function returns **XConverterNotFound**. For supported locales, existence of a converter from *COMPOUND_TEXT*, *STRING*, or the encoding of the current locale is guaranteed if **XSupportsLocale** returns True for the current locale (but the actual text may contain unconvertible characters). Conversion of other encodings to the encoding of the current locale is implementation dependent. In all of these error cases, the function does not set any return values.

If an element of the value member of *text_prop* is not convertible to **XmString**, the corresponding entry in the returned **XmStringTable** will be NULL, and **XmCvtTextPropertyToXmStringTable** returns Success.

To free the storage for the **XmStringTable** and its *count_return* compound strings returned by this function, first free each **XmString** in the table using **XmStringFree**, and then free the **XmStringTable** itself using **XtFree**.

display Specifies the connection to the X server.

text_prop Specifies a pointer to the *XTextProperty*. The format member of *text_prop* must be 8.

string_table_return
Specifies the **XmStringTable** array into which the converted compound strings are placed.

count_return Specifies the number of **XmString**s returned by this function.

Return Values

Upon success, this function returns the set of **XmString**s in *string_table_return*, and it returns the number of **XmString**s in *count_return*, and returns Success. Otherwise, it returns the following:

XLocaleNotSupported
Returned if conversion depends on the locale and the current locale is not supported.

XConverterNotFound
Returned if conversion to the encoding of the current locale is required and if the locale is supported but no converter is available for the encoding specified in *text_prop*.

Related Information

XmCvtXmStringTableToTextProperty(3), **XmText**(3), and **XmTextGetString**(3).

XmCvtXmStringTableToTextProperty

Purpose

A function that converts from XmStringTable to an XTextProperty Structure

Synopsis

#include <Xm/Xm.h>
int XmCvtXmStringTableToTextProperty (*display, string_table, count, style, text_prop_return***)**
    Display ****display*;**
    XmStringTable *string_table***;**
    int *count***;**
    XmICCEncodingStyle *style***;**
    XTextProperty *****text_prop_return*;**

Description

XmCvtXmStringTableToTextProperty converts the **XmString**s in the specified **XmStringTable** into an *XTextProperty* structure.

The function sets the encoding member of *text_prop_return* to an **Atom** for the specified display naming the encoding determined by the specified style, and it converts the first *count* compound strings in the specified **XmStringTable** to this encoding for storage in the *text_prop_return* value member. Following are the possible encoding styles:

XmSTYLE_COMPOUND_STRING
: The encoding is _MOTIF_COMPOUND_STRING. The function converts each specified **XmString** to a compound string in Byte Stream format.

XmSTYLE_COMPOUND_TEXT
: The encoding is *COMPOUND_TEXT*. The function converts each specified **XmString** to compound text.

XmSTYLE_LOCALE
: The encoding is the encoding of the current locale. The function converts each specified **XmString** to the encoding of the current locale.

XmSTYLE_STRING
The encoding is *STRING* (plain C strings encoded in ISO8859-1), and the function converts each specified **XmString** to *STRING*.

XmSTYLE_TEXT
If all specified **XmString**s are fully convertible to the encoding of the current locale, the encoding is the encoding of the current locale, and the function converts each specified **XmString** to the encoding of the current locale. Otherwise, the encoding is *COMPOUND_TEXT*, and the function converts each specified compound string to compound text.

XmSTYLE_STANDARD_ICC_TEXT
If all specified **XmString**s are fully convertible to *STRING*, the encoding is *STRING*, and the function converts each specified **XmString** to *STRING*. Otherwise, the encoding is *COMPOUND_TEXT*, and the function converts each specified **XmString** to compound text.

display Specifies the connection to the X server.

string_table Specifies a set of **XmString**s.

count Specifies the number of **XmString**s to be converted in *string_table*.

style Specifies the manner in which the property is encoded.

text_prop_return
Returns the *XTextProperty* structure.

To free the storage for the value member of the *XTextProperty*, use **XtFree**.

Return Values

If conversion depends on the locale and the current locale is not supported, the function returns **XLocaleNotSupported**. In both of these cases, the function does not set *text_prop_return*.

To determine whether the function is guaranteed not to return **XLocaleNotSupported**, use **XSupportsLocale**.

Related Information

XmCvtXmStringToByteStream(3), **XmCvtTextPropertyToXmStringTable**(3), and **XmStringTable**(3).

XmCvtXmStringToByteStream

Purpose

A compound string function that converts a compound string to a Byte Stream format

Synopsis

#include <Xm/Xm.h>

unsigned int XmCvtXmStringToByteStream(
　　XmString *string***,**
　　unsigned char *****prop_return*);**

Description

XmCvtXmStringToByteStream converts a compound string to a string of bytes representing the compound string in Byte Stream format. This routine is typically used by the source of a data transfer operation to produce a Byte Stream representation for transferring a compound string to a destination.

If *prop_return* is not NULL, this function creates a string of characters in Byte Stream format and returns it in *prop_return*. The function also returns the number of bytes in *prop_return*. If *prop_return* is NULL, the function does not return the Byte Stream format string, but it does calculate and return the number of bytes that would appear in the Byte Stream format string.

string	Specifies a compound string to be converted to Byte Stream format
prop_return	Specifies a pointer to a string in Byte Stream format that is created and returned by this function. If *prop_return* is NULL, no Byte Stream format string is returned. When a Byte Stream format string is returned, the function allocates space to hold it. The application is responsible for managing this allocated space. The application can recover the allocated space by calling **XtFree**.

Return Values

Returns the number of bytes in the Byte Stream representation (whether or not the Byte Stream representation is returned).

Related Information

XmString(3) and **XmCvtByteStreamToXmString**(3).

XmCvtXmStringToCT

Purpose

A compound string function that converts a compound string to compound text

Synopsis

#include <Xm/Xm.h>

char * XmCvtXmStringToCT(
XmString *string***);**

Description

XmCvtXmStringToCT converts a compound string to a (**char ***) string in compound text format. The application must call **XtAppInitialize** before calling this function. The converter uses the font list tag associated with a given compound string segment to select a compound text format for that segment. A registry defines a mapping between font list tags and compound text encoding formats. The converter uses the following algorithm for each compound string segment:

1. If the compound string segment tag is mapped to **XmFONTLIST_DEFAULT_TAG** in the registry, the converter passes the text of the compound string segment to **XmbTextListToTextProperty** with an encoding style of **XCompoundTextStyle** and uses the resulting compound text for that segment.
2. If the compound string segment tag is mapped to an MIT registered charset in the registry, the converter creates the compound text for that segment using the charset (from the registry) and the text of the compound string segment as defined in the X Consortium Standard *Compound Text Encoding*.
3. If the compound string segment tag is mapped to a charset in the registry that is neither **XmFONTLIST_DEFAULT_TAG** nor an MIT registered charset, the converter creates the compound text for that segment using the charset (from the registry) and the text of the compound string segment as an "extended segment" with a variable number of octets per character.
4. If the compound string segment tag is not mapped in the registry, the result is implementation dependent.

string Specifies a compound string to be converted to compound text.

Return Values

Returns a (**char ***) string in compound text format. This format is described in the X Consortium Standard *Compound Text Encoding*. The function allocates space to hold the returned string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XtFree**.

Related Information

XmCvtCTToXmString(3), **XmFontList**(3), **XmMapSegmentEncoding**(3), **XmRegisterSegmentEncoding**(3), and **XmString**.

XmDeactivateProtocol

Purpose

A VendorShell function that deactivates a protocol without removing it

Synopsis

```
#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmDeactivateProtocol(
        Widget shell,
        Atom property,
        Atom protocol);
```

Description

XmDeactivateProtocol deactivates a protocol without removing it. It updates the handlers and the *property* if the *shell* is realized. It is sometimes useful to allow a protocol's state information (callback lists, and so on) to persist, even though the client may choose to temporarily resign from the interaction. The main use of this capability is to gray/ungray **f.send_msg** entries in the MWM system menu. To support this capability, *protocol* is allowed to be in one of two states: active or inactive. If *protocol* is active and *shell* is realized, *property* contains the *protocol* **Atom**. If *protocol* is inactive, **Atom** is not present in the *property*.

XmDeactivateWMProtocol is a convenience interface. It calls **XmDeactivateProtocol** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocol	Specifies the protocol atom

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

mwm(1), **VendorShell**(3), **XmActivateProtocol**(3), **XmDeactivateWMProtocol**(3), and **XmInternAtom**(3).

XmDeactivateWMProtocol

Purpose

A VendorShell convenience interface that deactivates a protocol without removing it

Synopsis

```
#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmDeactivateWMProtocol(
        Widget shell,
        Atom protocol);
```

Description

XmDeactivateWMProtocol is a convenience interface. It calls **XmDeactivateProtocol** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
protocol	Specifies the protocol atom

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmActivateWMProtocol**(3), **XmDeactivateProtocol**(3), and **XmInternAtom**(3).

XmDestroyPixmap

Purpose

A pixmap caching function that removes a pixmap from the pixmap cache

Synopsis

#include <Xm/Xm.h>

Boolean XmDestroyPixmap(
  Screen * *screen*,
  Pixmap *pixmap***);**

Description

XmDestroyPixmap removes pixmaps that are no longer used. Pixmaps are completely freed only when there is no further reference to them.

screen	Specifies the display screen for which the pixmap was requested
pixmap	Specifies the pixmap to be destroyed

Return Values

Returns True when successful; returns False if there is no matching screen and pixmap in the pixmap cache.

Related Information

XmInstallImage(3), **XmUninstallImage**(3), and **XmGetPixmap**(3).

XmDirectionMatch

Purpose

A function that checks for a specified direction component

Synopsis

#include <Xm/Xm.h>
Boolean XmDirectionMatch (*d1*, *d2***)**
    XmDirection *d1***;**
    XmDirection *d2***;**

Description

XmDirectionMatch compares two **XmDirection** values. The function returns a Boolean value depending on whether or not the two input values "match." The simplest match is when *d1* and *d2* are identical. However, other matches are possible. **XmDirectionMatch** attempts to compare specified bits only; nonspecified bits automatically match.

For example, suppose that *d1* equals **XmTOP_TO_BOTTOM_RIGHT_TO_LEFT**. In this case, the function will return True if *d2* equals either **XmRIGHT_TO_LEFT** or **XmTOP_TO_BOTTOM**. However, the function will return False if *d2* equals **XmTOP_TO_BOTTOM_LEFT_TO_RIGHT**, **XmBOTTOM_TO_TOP_RIGHT_TO_LEFT**, or **XmBOTTOM_TO_TOP_LEFT_TO_RIGHT**.

Note that direction can be thought of as having three components, a horizontal component, a vertical component, and the precedence among them. This means that in addition to the previously mentioned directions, the function will still return False if *d1* equals **XmTOP_TO_BOTTOM_RIGHT_TO_LEFT** and *d2* equals **XmRIGHT_TO_LEFT_TOP_TO_BOTTOM**.

d1	Specifies an **XmDirection** value.
d2	Specifies an **XmDirection** value.

Return Values

Returns True if *d1* "matches" *d2*; otherwise, returns False.

Related Information

XmDirection(3), **XmDirectionMatchPartial**(3), **XmDirectionToStringDirection**(3), **XmString**(3), **XmStringDirection**(3), and **XmStringDirectionToDirection**(3).

XmDirectionMatchPartial

Purpose

A function that checks for a specified direction component

Synopsis

#include <Xm/Xm.h>
Boolean XmDirectionMatchPartial (*d1*, *d2*, *dmask***)**
    XmDirection *d1***;**
    XmDirection *d2***;**
    XmDirection *dmask***;**

Description

XmDirectionMatchPartial compares *d1* and *d2* along the direction component specified by *dmask*. For example, if *dmask* equals **XmVERTICAL_MASK**, then the function will compare only the vertical components of *d1* and *d2*.

d1 Specifies an **XmDirection** value to check.

d2 Specifies an **XmDirection** value to check.

dmask Specifies the direction component along which *d1* and *d2* are to be checked. Appropriate values for *dmask* are **XmHORIZONTAL_MASK**, **XmVERTICAL_MASK**, and **XmPRECEDENCE_MASK**.

Return Values

Returns True if the *d1* and *d2* match in the component specified by *dmask*; otherwise, returns False.

Related Information

XmDirection(3), **XmDirectionMatch**(3), **XmDirectionToStringDirection**(3), **XmStringDirection**(3), and **XmStringDirectionToDirection**(3).

XmDirectionToStringDirection

Purpose

A function that converts an XmDirection value to an XmStringDirection value

Synopsis

#include <Xm/Xm.h>
XmStringDirection XmDirectionToStringDirection (*dir***)**
        XmDirection   *dir***;**

Description

XmDirectionToStringDirection converts the specified **XmDirection** direction value to its equivalent **XmStringDirection** value. Basically, if the **XmDirection** value has a horizontal direction specification, that horizontal element is used; otherwise, the **XmStringDirection** value is interpreted as **XmSTRING_DIRECTION_L_TO_R**. This function provides backward compatibility with the **XmStringDirection** data type.

Note that the Motif toolkit also contains an **XmStringDirectionToDirection** routine to convert an **XmStringDirection** value to its **XmDirection** equivalent.

dir   Specifies the **XmDirection** value to be converted.

Return Values

Returns the following **XmStringDirection** values:

XmSTRING_DIRECTION_R_TO_L
If the *dir* argument has a right to left horizontal direction value in it, for example **XmRIGHT_TO_LEFT_TOP_TO_BOTTOM**.

XmSTRING_DIRECTION_L_TO_R
If the *dir* argument has a left to right horizontal direction in it, for example **XmLEFT_TO_RIGHT_TOP_TO_BOTTOM**, or if the horizontal direction value in the *dir* argument is ambiguous, such as in the **XmTOP_TO_BOTTOM** value.

XmSTRING_DIRECTION_DEFAULT
If there was no horizontal direction specified.

Related Information

XmDirection(3), **XmDirectionMatch**(3), **XmDirectionMatchPartial**(3), **XmDirectionToStringDirection**(3), **XmString**(3), **XmStringDirection**(3), and **XmStringDirectionToDirection**(3),

XmDragCancel

Purpose

A Drag and Drop function that terminates a drag transaction

Synopsis

#include <Xm/DragDrop.h>

void XmDragCancel(
    Widget *dragcontext***);**

Description

XmDragCancel terminates a drag operation and cancels any pending actions of the specified DragContext. This routine can only be called by the initiator client.

dragcontext Specifies the ID of the DragContext widget associated with the drag and drop transaction to be terminated

For a complete definition of DragContext and its associated resources, see **XmDragContext**(3).

Related Information

XmDragContext(3) and **XmDragStart**(3).

XmDragStart

Purpose

A Drag and Drop function that initiates a drag and drop transaction

Synopsis

#include <Xm/DragDrop.h>

Widget XmDragStart(
  Widget *widget***,**
  XEvent ****event*,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmDragStart initiates a drag operation. This routine returns the DragContext widget that it initializes for the associated drag transaction. The toolkit is responsible for freeing the DragContext when the drag and drop transaction is complete.

widget	Specifies the ID of the smallest widget and/or gadget that encloses the source elements selected for a drag operation.
event	Specifies the *XEvent* that triggered the drag operation. This event must be a ButtonPress event.
arglist	Specifies the argument list. Any **XmDragContext** resources not specified in the argument list are obtained from the resource database or are set to their default values.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DragContext and its associated resources, see **XmDragContext**(3).

Return Values

Returns the ID of the DragContext widget that controls this drag and drop transaction. Returns NULL if the drag cannot be initiated.

Related Information

XmDragCancel(3) and **XmDragContext**(3).

XmDropSite

Purpose

The DropSite Registry

Synopsis

#include <Xm/DragDrop.h>

Description

A client registers a widget or gadget as a drop site using the **XmDropSiteRegister** function. In addition, this routine defines the behavior and capabilities of a drop site by specifying appropriate resources. For example, the **XmNimportTargets** and **XmNnumImportTargets** resources identify respectively the selection target types and number of types supported by a drop site. The visual animation effects associated with a drop site are also described with DropSite resources.

Drop site animation effects that occur in response to the pointer entering a valid drop site are called drag-under effects. A receiver can select from several animation styles supplied by the toolkit or can provide customized animation effects. Drag-under effects supplied by the toolkit include border highlighting, shadow in/out drawing, and pixmap representation.

When a preregister drag protocol style is used, the toolkit generates drag-under visual effects based on the value of the **XmNanimationStyle** resource. In dynamic mode, if the drop site **XmNdragProc** resource is NULL, the toolkit also provides animation effects based on the **XmNanimationStyle** resource. Otherwise, if the **XmNdragProc** routine is specified, the receiver can either assume responsibility for animation effects (through the **XmNdragProc** routine) or rely on the toolkit to provide animation. An application can either handle all or none of the animation effects for a particular drop site. That is, an application should never do a partial job of animation on a particular drop site.

Drop sites may overlap. The initial stacking order corresponds to the order in which the drop sites were registered. When a drop site overlaps another drop site, the drag-under effects of the drop site underneath are clipped by the obscuring drop site(s).

The **XmDropSiteUpdate** routine sets resources for a widget that is registered as a drop site. **XmDropSiteRetrieve** gets drop site resource values previously specified for a registered widget. These routines are used instead of **XtSetValues** and **XtGetValues**.

Classes

XmDropSite does not inherit from any widget class.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XmDropSiteUpdate** (S), retrieved by using **XmDropSiteRetrieve** (G), or is not applicable (N/A).

XmDropSite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNanimationMask	XmCAnimationMask	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNanimationPixmap	XmCAnimationPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNanimationPixmap-Depth	XmCAnimationPixmap-Depth	int	0	CSG
XmNanimationStyle	XmCAnimationStyle	unsigned char	XmDRAG_UNDER_-HIGHLIGHT	CSG
XmNdragProc	XmCDragProc	XtCallbackProc	NULL	CSG
XmNdropProc	XmCDropProc	XtCallbackProc	NULL	CSG
XmNdropRectangles	XmCDropRectangles	XRectangle *	dynamic	CSG
XmNdropSiteActivity	XmCDropSite- Activity	unsigned char	XmDROP_SITE_-ACTIVE	CSG
XmNdropSiteOperations	XmCDropSite-Operations	unsigned char	XmDROP_MOVE | -XmDROP_COPY	CSG
XmNdropSiteType	XmCDropSiteType	unsigned char	XmDROP_SITE_-SIMPLE	CG

XmNimportTargets	XmCImportTargets	Atom *	NULL	CSG
XmNnumDropRectangles	XmCNumDrop-Rectangles	Cardinal	1	CSG
XmNnumImportTargets	XmCNumImport-Targets	Cardinal	0	CSG

XmNanimationMask

Specifies a mask to use with the pixmap specified by **XmNanimationPixmap** when the animation style is **XmDRAG_UNDER_PIXMAP**.

XmNanimationPixmap

Specifies a pixmap for drag-under animation when the animation style is **XmDRAG_UNDER_PIXMAP**. The pixmap is drawn with its origin at the upper left corner of the bounding box of the drop site. If the drop site window is larger than the animation pixmap, the portion of the window not covered by the pixmap will be tiled with the window's background color.

XmNanimationPixmapDepth

Specifies the depth of the pixmap specified by the **XmNanimationPixmap** resource. When the depth is 1, the colors are taken from the foreground and background of the drop site widget. For any other value, drop site animation occurs only if the **XmNanimationPixmapDepth** matches the depth of the drop site window. Colors are derived from the current colormap.

XmNanimationStyle

Specifies the drag-under animation style used when a drag enters a valid drop site. The possible values are

XmDRAG_UNDER_HIGHLIGHT

The drop site uses highlighting effects.

XmDRAG_UNDER_SHADOW_OUT

The drop site uses an outset shadow.

XmDRAG_UNDER_SHADOW_IN

The drop site uses an inset shadow.

XmDRAG_UNDER_PIXMAP
The drop site uses the pixmap specified by **XmNanimationPixmap** to indicate that it can receive the drop.

XmDRAG_UNDER_NONE
The drop site does not use animation effects. A client using a dynamic protocol, may provide drag-under effects in its **XmNdragProc** routine.

XmNdragProc
Specifies the procedure that is invoked when the drop site receives a crossing, motion, or operation changed message. This procedure is called only when a dynamic protocol is used. The type of structure whose address is passed to this procedure is **XmDragProcCallbackStruct**. The reason sent to the procedure is one of the following:

- **XmCR_DROP_SITE_ENTER_MESSAGE**
- **XmCR_DROP_SITE_LEAVE_MESSAGE**
- **XmCR_DRAG_MOTION**
- **XmCR_OPERATION_CHANGED**

The drag procedure may change the values of some members of the **XmDragProcCallbackStruct** passed to it. After the drag procedure returns, the toolkit uses the final values in initializing some members of the callback structure passed to the appropriate callbacks of the initiator (the DragContext's **XmNdropSiteEnterCallback**, **XmNdropSiteLeaveCallback**, **XmNdragMotionCallback**, or **XmNoperationChangedCallback** callbacks).

XmNdropProc
Specifies the procedure that is invoked when a drop (excluding a cancel or interrupt action) occurs on a drop site regardless of the status of the drop site. The type of the structure whose address is passed to this procedure is **XmDropProcCallbackStruct**. The reason sent to the procedure is **XmCR_DROP_MESSAGE**.

The drop procedure may change the values of some members of the **XmDropProcCallbackStruct** passed to it. After the drop procedure returns, the toolkit uses the final values in initializing some members of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

XmNdropRectangles

Specifies a list of rectangles that describe the shape of a drop site. The locations of the rectangles are relative to the origin of the enclosing object. When **XmNdropRectangles** is NULL, the drop site is assumed to be the sensitive area of the enclosing widget. If **XmNdropSiteType** is **XmDROP_SITE_COMPOSITE**, this resource cannot be specified by the application.

Retrieving this resource returns allocated memory that needs to be freed with the **XtFree** function.

XmNdropSiteActivity

Indicates whether a drop site is active or inactive. The values are **XmDROP_SITE_ACTIVE**, **XmDROP_SITE_INACTIVE**, and **XmDROP_SITE_IGNORE**. An active drop site can receive a drop, whereas an inactive drop site is dormant. An inactive drop site is treated as if it was not a registered drop site and any drag-under visuals associated with entering or leaving the drop site do not occur. However, it is still used for clipping drag-under effects. A value of **XmDROP_SITE_IGNORE** indicates that a drop site should be ignored for all purposes.

XmNdropSiteOperations

Specifies the set of valid operations associated with a drop site. This resource is a bit mask that is formed by combining one or more of the following values using a bitwise operation such as inclusive OR (|): **XmDROP_COPY**, **XmDROP_LINK**, and **XmDROP_MOVE**. The value **XmDROP_NOOP** for this resource indicates that no operations are valid.

XmNdropSiteType

Specifies the type of the drop site. The possible values are

XmDROP_SITE_SIMPLE

The widget does not have any additional children that are registered as drop sites.

XmDROP_SITE_COMPOSITE

The widget will have children that are registered as drop sites.

XmNimportTargets
Specifies the list of target atoms that this drop site accepts.

XmNnumDropRectangles
Specifies the number of rectangles in the **XmNdropRectangles** list. If the drop site type is **XmDROP_SITE_COMPOSITE**, this resource can not be specified by the application.

XmNnumImportTargets
Specifies the number of atoms in the target atom list.

Callback Information

A pointer to the following structure is passed to the **XmNdragProc** routine when the drop site receives crossing, motion, or operation changed messages:

```
typedef struct
{
        int reason;
        XEvent *event;
        Time timeStamp;
        Widget dragContext;
        Position x;
        Position y;
        unsigned char dropSiteStatus;
        unsigned char operation;
        unsigned char operations;
        Boolean animate;
} XmDragProcCallbackStruct, *XmDragProcCallback;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

dragContext Specifies the ID of the DragContext widget associated with the transaction.

x Indicates the x-coordinate of the pointer relative to the drop site.

y Indicates the y-coordinate of the pointer relative to the drop site.

dropSiteStatus
An IN/OUT member that indicates whether or not a drop site is valid.

When *reason* is **XmCR_DROP_SITE_ENTER_MESSAGE** or **XmCR_OPERATION_CHANGED**, or *reason* is **XmCR_DRAG_MOTION** or **XmCR_DROP_SITE_LEAVE_MESSAGE** and the pointer is not in the same drop site as on the previous invocation of the drag procedure, the toolkit initializes **dropSiteStatus** to **XmDROP_SITE_VALID** if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the *operation* member is not **XmDROP_NOOP**. Otherwise, the toolkit initializes **dropSiteStatus** to **XmDROP_SITE_INVALID**.

When the *reason* is **XmCR_DRAG_MOTION** or **XmCR_DROP_SITE_LEAVE_MESSAGE** and the pointer is within the same drop site as on the previous invocation of the drag procedure, the toolkit initializes **dropSiteStatus** to the value of **dropSiteStatus** at the time the previous invocation of the drag procedure returns.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the **dropSiteStatus** member of the callback struct passed to the appropriate callbacks of the initiator.

operation An IN/OUT member that identifies an operation.

The toolkit initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the *operation* member of the callback struct passed to the appropriate callbacks of the initiator.

operations An IN/OUT member that indicates the set of operations supported for the source data.

If the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource. If the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the *operations* member of the callback struct passed to the appropriate callbacks of the initiator.

animate An OUT member that indicates whether the toolkit or the receiver client provides drag-under effects for a valid drop site. If *animate* is set to True, the toolkit provides drop site animation per the **XmNanimationStyle** resource value; if it is set to False, the receiver generates drag-under animation effects.

A pointer to the following structure is passed to the **XmNdropProc** routine when the drop site receives a drop message:

```
typedef struct
{
        int reason;
        XEvent *event;
        Time timeStamp;
        Widget dragContext;
        Position x;
        Position y;
        unsigned char dropSiteStatus;
        unsigned char operation;
        unsigned char operations;
        unsigned char dropAction;
} XmDropProcCallbackStruct, *XmDropProcCallback;
```

reason Indicates why the callback was invoked.

event Specifies the *XEvent* that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

dragContext Specifies the ID of the DragContext widget associated with the transaction.

x Indicates the x-coordinate of the pointer relative to the drop site.

y Indicates the y-coordinate of the pointer relative to the drop site.

dropSiteStatus
An IN/OUT member that indicates whether or not a drop site is valid.

The toolkit initializes **dropSiteStatus** to **XmDROP_SITE_VALID** if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the *operation* member is not **XmDROP_NOOP**. Otherwise, the toolkit initializes **dropSiteStatus** to **XmDROP_SITE_INVALID**.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the **dropSiteStatus** member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

operation An IN/OUT member that identifies an operation.

The toolkit initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If it finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *operation* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

operations An IN/OUT member that indicates the set of operations supported for the source data.

If the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource. If the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *operations* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

dropAction An IN/OUT member that identifies the action associated with the drop. The possible values are

XmDROP A drop was attempted. If the drop site is valid, drop transfer handling proceeds.

XmDROP_HELP
The user has requested help on the drop site.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the **dropAction** member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

Related Information

XmDragContext(3), **XmDragIcon**(3), **XmDropSiteConfigureStackingOrder**(3), **XmDropSiteEndUpdate**(3), **XmDropSiteQueryStackingOrder**(3), **XmDropSiteRegister**(3), **XmDropSiteStartUpdate**(3), **XmDropSiteUpdate**(3), **XmDropSiteUnregister**(3), **XmDropTransfer**(3), and **XmTargetsAreCompatible**(3).

XmDropSiteConfigureStackingOrder

Purpose

A Drag and Drop function that reorders a stack of widgets that are registered drop sites

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteConfigureStackingOrder(
    Widget *widget*,
    Widget *sibling*,
    Cardinal *stack_mode*);

Description

XmDropSiteConfigureStackingOrder changes the stacking order of the drop site specified by *widget*. The stacking order controls the manner in which drag-under effects are clipped by overlapping siblings, regardless of whether they are active. The stack mode is relative either to the entire stack, or to another drop site within the stack. The stack order can be modified only if the drop sites are siblings in both the widget and drop site hierarchy, and the widget parent of the drop sites is registered as a composite drop site.

widget	Specifies the drop site to be restacked.
sibling	Specifies a sibling drop site for stacking operations. If specified, then *widget* is restacked relative to this drop site's stack position.
stack_mode	Specifies the new stack position for the specified widget. The values are **XmABOVE** and **XmBELOW**. If a sibling is specified, then *widget* is restacked as follows:

XmABOVE	The widget is placed just above the sibling.
XmBELOW	The widget is placed just below the sibling.

If the *sibling* parameter is not specified, then *widget* is restacked as follows:

XmABOVE The widget is placed at the top of the stack.

XmBELOW
The widget is placed at the bottom of the stack.

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDropSite(3), **XmDropSiteRetrieve**(3), and
XmDropSiteQueryStackingOrder(3).

XmDropSiteEndUpdate

Purpose

A Drag and Drop function that facilitates processing updates to multiple drop sites

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteEndUpdate(
  Widget *widget***);**

Description

XmDropSiteEndUpdate is used in conjunction with **XmDropSiteStartUpdate** to process updates to multiple drop sites within the same hierarchy. **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** signal the beginning and the end respectively of a series of calls to **XmDropSiteUpdate**. Calls to **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** can be recursively stacked. Using these routines optimizes the processing of update information.

widget Specifies the ID of any widget within a given hierarchy. The function uses this widget to identify the shell that contains the drop sites.

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDropSiteStartUpdate(3) and **XmDropSiteUpdate**(3).

XmDropSiteQueryStackingOrder

Purpose

A Drag and Drop function that returns the parent, a list of children, and the number of children for a specified widget

Synopsis

#include <Xm/DragDrop.h>

Status XmDropSiteQueryStackingOrder(
Widget *widget***,**
Widget ****parent_return*,**
Widget *****child_returns*,**
Cardinal ****num_child_returns*);**

Description

XmDropSiteQueryStackingOrder obtains the parent, a list of children registered as drop sites, and the number of children registered as drop sites for a given widget. The children are listed in current stacking order, from bottom-most (first child) to the top-most (last child). This function allocates memory for the returned data that must be freed by calling **XtFree**.

widget Specifies the widget ID. For this widget, you obtain the list of its children, its parent, and the number of children.

parent_return
Returns the widget ID of the drop site parent of the specified widget.

child_returns
Returns a pointer to the list of drop site children associated with the specified widget. The function allocates memory to hold the list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XtFree**.

num_child_returns
Returns the number of drop site children for the specified widget.

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Return Values

Returns 0 (zero) if the routine fails; returns a nonzero value if it succeeds.

Related Information

XmDropSite(3) and **XmDropSiteConfigureStackingOrder**(3).

XmDropSiteRegister

Purpose

A Drag and Drop function that identifies a drop site and assigns resources that specify its behavior

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteRegister(
    Widget *widget***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmDropSiteRegister identifies the specified widget or gadget as a drop site and sets resource values that define the drop site's behavior. The routine assigns default values to any resources that are not specified in the argument list. The toolkit generates a warning message if a drop site is registered with **XmNdropSiteActivity** set to **XmDROP_SITE_ACTIVE** and the **XmNdropProc** resource is NULL.

If the drop site is a descendant of a widget that is registered as a drop site, the **XmNdropSiteType** resource of the ancestor drop site must be specified as **XmDROP_SITE_COMPOSITE**. The ancestor must be registered before the descendant. The drop site is stacked above all other sibling drop sites already registered.

widget	Specifies the ID of the widget to be registered.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDisplay(3), **XmDropSite**(3), **XmDropSiteEndUpdate**(3), **XmDropSiteStartUpdate**(3), **XmDropSiteUpdate**(3), **XmDropSiteUnregister**(3), and **XmScreen**(3).

XmDropSiteRegistered

Purpose

A Drag and Drop function that determines if a drop site has been registered

Synopsis

#include <Xm/DragDrop.h>

Boolean XmDropSiteRegistered(
    Widget *widget***);**

Description

XmDropSiteRegistered determines if the specified widget has a drop site registered. If a drop site is registered, this function returns True.

widget	Specifies the ID of the widget being queried.

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Return Values

If the widget is not a registered drop site, this function returns False. Otherwise, it returns True.

Related Information

XmDisplay(3), **XmDropSite**(3), **XmDropSiteEndUpdate**(3), **XmDropSiteStartUpdate**(3), **XmDropSiteUpdate**(3), **XmDropSiteUnregister**(3), and **XmScreen**(3).

XmDropSiteRetrieve

Purpose

A Drag and Drop function that retrieves resource values set on a drop site

Synopsis

```
#include <Xm/DragDrop.h>

void XmDropSiteRetrieve(
        Widget widget,
        ArgList arglist,
        Cardinal argcount);
```

Description

XmDropSiteRetrieve extracts values for the given resources from the drop site specified by *widget*. An initiator can also obtain information about the current drop site by passing the associated DragContext widget as the *widget* parameter to this routine. The initiator can retrieve all of the drop site resources except **XmNdragProc** and **XmNdropProc** using this method.

widget	Specifies the ID of the widget that encloses the drop site.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDropSite(3) and **XmDropSiteUpdate**(3).

XmDropSiteStartUpdate

Purpose

A Drag and Drop function that facilitates processing updates to multiple drop sites

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteStartUpdate(
Widget *widget***);**

Description

XmDropSiteStartUpdate is used in conjunction with **XmDropSiteEndUpdate** to process updates to multiple drop sites within the same shell widget. **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** signal the beginning and the end respectively of a series of calls to **XmDropSiteUpdate**. Calls to **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** can be recursively stacked. Using these routines optimizes the processing of update information.

widget Specifies the ID of any widget within a given hierarchy. The function uses this widget to identify the shell that contains the drop sites.

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDropSite(3), **XmDropSiteEndUpdate**(3), and **XmDropSiteUpdate**(3).

XmDropSiteUnregister

Purpose

A Drag and Drop function that frees drop site information

Synopsis

```
#include <Xm/DragDrop.h>

void XmDropSiteUnregister(
        Widget widget);
```

Description

XmDropSiteUnregister informs the toolkit that the specified widget is no longer a registered drop site. The function frees all associated drop site information.

widget	Specifies the ID of the widget, registered as a drop site, that is to be unregistered

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDropSite(3) and **XmDropSiteRegister**(3).

XmDropSiteUpdate

Purpose

A Drag and Drop function that sets resource values for a drop site

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteUpdate(
    Widget *widget***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmDropSiteUpdate modifies drop site resources associated with the specified widget. This routine updates the drop site resources specified in the *arglist*.

widget	Specifies the ID of the widget registered as a drop site
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DropSite and its associated resources, see **XmDropSite**(3).

Related Information

XmDropSite(3), **XmDropSiteEndUpdate**(3), **XmDropSiteRegister**(3), **XmDropSiteStartUpdate**(3), and **XmDropSiteUnregister**(3).

XmDropTransferAdd

Purpose

A Drag and Drop function that enables additional drop transfer entries to be processed after initiating a drop transfer

Synopsis

#include <Xm/DragDrop.h>

void XmDropTransferAdd(
    Widget *drop_transfer***,**
    XmDropTransferEntryRec ***transfers*,**
    Cardinal *num_transfers***);**

Description

XmDropTransferAdd identifies a list of additional drop transfer entries to be processed after a drop transfer is started.

drop_transfer
: Specifies the ID of the DropTransfer widget returned by **XmDropTransferStart**

transfers
: Specifies the additional drop transfer entries that the receiver wants processed

num_transfers
: Specifies the number of items in the *transfers* array

For a complete definition of DropTransfer and its associated resources, see **XmDropTransfer**(3).

Related Information

XmDragContext(3), **XmDropTransfer**(3), and **XmDropTransferStart**(3).

XmDropTransferStart

Purpose

A Drag and Drop function that initiates a drop transfer

Synopsis

#include <Xm/DragDrop.h>

Widget XmDropTransferStart(
    Widget *widget***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmDropTransferStart initiates a drop transfer and uses the specified argument list to initialize an **XmDropTransfer** object. The DropTransfer object can be manipulated with **XtSetValues** and **XtGetValues** until the last call to the **XmNtransferProc** procedure is made. After that point, the result of using the widget pointer is undefined. The DropTransfer object is freed by the toolkit when a transfer is complete.

widget	Specifies the ID of the DragContext widget associated with the transaction
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DropTransfer and its associated resources, see **XmDropTransfer**(3).

Return Values

Returns the ID of the DropTransfer widget.

Related Information

XmDragContext(3), **XmDropTransfer**(3), and **XmDropTransferAdd**(3).

XmFileSelectionBoxGetChild

Purpose

A FileSelectionBox function used to access a component

Synopsis

```
#include <Xm/FileSB.h>

Widget XmFileSelectionBoxGetChild(
        Widget widget,
        unsigned char child);
```

Description

XmFileSelectionBoxGetChild is used to access a component within a FileSelectionBox. The parameters given to the function are the FileSelectionBox widget and a value indicating which component to access.

NOTE: This routine is obsolete and exists for compatibility with previous releases. Instead of calling **XmFileSelectionBoxGetChild**, you should call **XtNameToWidget** as described in the **XmFileSelectionBox**(3) reference page.

widget Specifies the FileSelectionBox widget ID.

child Specifies a component within the FileSelectionBox. The following are legal values for this parameter:

- **XmDIALOG_APPLY_BUTTON**
- **XmDIALOG_CANCEL_BUTTON**
- **XmDIALOG_DEFAULT_BUTTON**
- **XmDIALOG_DIR_LIST**
- **XmDIALOG_DIR_LIST_LABEL**
- **XmDIALOG_FILTER_LABEL**
- **XmDIALOG_FILTER_TEXT**
- **XmDIALOG_HELP_BUTTON**

- **XmDIALOG_LIST**
- **XmDIALOG_LIST_LABEL**
- **XmDIALOG_OK_BUTTON**
- **XmDIALOG_SELECTION_LABEL**
- **XmDIALOG_SEPARATOR**
- **XmDIALOG_TEXT**
- **XmDIALOG_WORK_AREA**

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox**(3).

Return Values

Returns the widget ID of the specified FileSelectionBox component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmFileSelectionBox(3).

XmFileSelectionDoSearch

Purpose

A FileSelectionBox function that initiates a directory search

Synopsis

#include <Xm/FileSB.h>

void XmFileSelectionDoSearch(
  Widget *widget***,**
  XmString *dirmask***);**

Description

XmFileSelectionDoSearch initiates a directory and file search in a FileSelectionBox widget. For a description of the actions that the FileSelectionBox takes when doing a search, see **XmFileSelectionBox**(3).

widget	Specifies the FileSelectionBox widget ID.
dirmask	Specifies the directory mask used in determining the directories and files displayed in the FileSelectionBox lists. This value is used as the *mask* member of the input data **XmFileSelectionBoxCallbackStruct** structure passed to the FileSelectionBox's **XmNqualifySearchDataProc**. The *dir* and *pattern* members of that structure are NULL.

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox**(3).

Related Information

XmFileSelectionBox(3).

XmFontListAdd

Purpose

A font list function that creates a new font list

Synopsis

#include <Xm/Xm.h>

XmFontList XmFontListAdd(
  XmFontList *oldlist***,**
  XFontStruct **font***,**
  XmStringCharSet *charset***);**

Description

XmFontListAdd creates a new font list consisting of the contents of *oldlist* and the new font list element being added. This function deallocates *oldlist* after extracting the required information; therefore, do not reference *oldlist* thereafter.

NOTE: This function is obsolete and exists for compatibility with previous releases. It has been replaced by **XmFontListAppendEntry**.

oldlist	Specifies a pointer to the font list to which an entry will be added.
font	Specifies a pointer to a font structure for which the new font list is generated. This is the structure returned by the XLib **XLoadQueryFont** function.
charset	Specifies the character set identifier for the font. This can be **XmSTRING_DEFAULT_CHARSET**, but this value does not comply with the AES, and it may be removed in future versions of Motif. If the value is **XmSTRING_DEFAULT_CHARSET**, the routine derives the character set from the current language environment.

Return Values

Returns NULL if *oldlist* is NULL; returns *oldlist* if *font* or *charset* is NULL; otherwise, returns a new font list.

Related Information

XmFontList(3) and **XmFontListAppendEntry**(3).

XmFontListAppendEntry

Purpose

A font list function that appends an entry to a font list

Synopsis

#include <Xm/Xm.h>

XmFontList XmFontListAppendEntry(
XmFontList *oldlist***,**
XmFontListEntry *entry***);**

Description

XmFontListAppendEntry creates a new font list that contains the contents of *oldlist*. This function copies the contents of the font list entry being added into this new font list. If *oldlist* is NULL, **XmFontListAppendEntry** creates a new font list containing only the single entry specified.

This function deallocates the original font list after extracting the required information. The caller must free the font list entry by using **XmFontListEntryFree**.

oldlist	Specifies the font list to be added to
entry	Specifies the font list entry to be added

Return Values

If *entry* is NULL, returns *oldlist*; otherwise, returns a new font list.

Related Information

XmFontList(3), **XmFontListEntryCreate**(3), **XmFontListEntryFree**(3), **XmFontListEntryLoad**(3), **XmFontListFree**(3), and **XmFontListRemoveEntry**(3).

XmFontListCopy

Purpose

A font list function that copies a font list

Synopsis

#include <Xm/Xm.h>

```
XmFontList XmFontListCopy(
        XmFontList fontlist);
```

Description

XmFontListCopy creates a new font list consisting of the contents of the *fontlist* argument.

fontlist	Specifies a font list to be copied

Return Values

Returns NULL if *fontlist* is NULL; otherwise, returns a new font list and allocates space to hold the font list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmFontListFree**.

Related Information

XmFontList(3) and **XmFontListFree**(3).

XmFontListCreate

Purpose

A font list function that creates a font list

Synopsis

#include <Xm/Xm.h>

XmFontList XmFontListCreate(
  XFontStruct * *font*,
  XmStringCharSet *charset*);

Description

XmFontListCreate creates a new font list with a single element specified by the provided font and character set. It also allocates the space for the font list.

NOTE: This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmFontListAppendEntry**.

font	Specifies a pointer to a font structure for which the new font list is generated. This is the structure returned by the XLib **XLoadQueryFont** function.
charset	Specifies the character set identifier for the font. This can be **XmSTRING_DEFAULT_CHARSET**, but this value does not comply with the AES, and it may be removed in future versions of Motif. If the value is **XmSTRING_DEFAULT_CHARSET**, the routine derives the character set from the current language environment.

Return Values

Returns NULL if *font* or *charset* is NULL; otherwise, returns a new font list.

Related Information

XmFontList(3) and **XmFontListAppendEntry**(3).

XmFontListEntryCreate

Purpose

A font list function that creates a font list entry

Synopsis

#include <Xm/Xm.h>

XmFontListEntry XmFontListEntryCreate(
char ****tag*,**
XmFontType *type***,**
XtPointer *font***);**

Description

XmFontListEntryCreate creates a font list entry that contains either a font or font set and is identified by a tag.

tag	Specifies a NULL terminated string for the tag of the font list entry. The tag may be specified as **XmFONTLIST_DEFAULT_TAG**, which is used to identify the default font list element in a font list.
type	Specifies whether the *font* argument is a font structure or a font set. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.
font	Specifies either an *XFontSet* returned by *XCreateFontSet* or a pointer to an *XFontStruct* returned by **XLoadQueryFont**.

The toolkit does not copy the X Font structure specified by the *font* argument. Therefore, an application programmer must not free *XFontStruct* or *XFontSet* until all font lists and/or font entries that reference it have been freed.

Return Values

Returns a font list entry. The function allocates space to hold the returned font list entry. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmFontListEntryFree**.

Related Information

XmFontList(3), **XmFontListAppendEntry**(3), **XmFontListEntryFree**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryGetTag**(3), **XmFontListEntryLoad**(3), and **XmFontListRemoveEntry**(3).

XmFontListEntryFree

Purpose

A font list function that recovers memory used by a font list entry

Synopsis

#include <Xm/Xm.h>

void XmFontListEntryFree(
XmFontListEntry ****entry*);**

Description

XmFontListEntryFree recovers memory used by a font list entry. This routine does not free the *XFontSet* or *XFontStruct* associated with the font list entry.

entry Specifies a pointer to the font list entry to be freed. In addition, it may be necessary to take the address of the font list entry (via the **&** operator) before passing it to this function.

Related Information

XmFontList(3), **XmFontListAppendEntry**(3), **XmFontListEntryCreate**(3), **XmFontListEntryLoad**(3), **XmFontListNextEntry**(3), and **XmFontListRemoveEntry**(3).

XmFontListEntryGetFont

Purpose

A font list function that retrieves font information from a font list entry

Synopsis

#include <Xm/Xm.h>

XtPointer XmFontListEntryGetFont(
    XmFontListEntry *entry***,**
    XmFontType ****type_return*);**

Description

XmFontListEntryGetFont retrieves font information for a specified font list entry. If the font list entry contains a font, *type_return* returns **XmFONT_IS_FONT** and the function returns a pointer to an *XFontStruct*. If the font list entry contains a font set, *type_return* returns **XmFONT_IS_FONTSET** and the function returns the *XFontSet*.

entry	Specifies the font list entry.
type_return	Specifies a pointer to the type of the font element for the current entry. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.

The returned *XFontSet* or *XFontStruct* is not a copy of the toolkit data and must not be freed.

Return Values

Returns an *XFontSet* or a pointer to an *XFontStruct* structure.

Related Information

XmFontList(3), **XmFontListEntryCreate**(3), **XmFontListEntryGetTag**(3) **XmFontListEntryLoad**(3), and **XmFontListNextEntry**(3).

XmFontListEntryGetTag

Purpose

A font list function that retrieves the tag of a font list entry

Synopsis

#include <Xm/Xm.h>

```
char* XmFontListEntryGetTag(
        XmFontListEntry entry);
```

Description

XmFontListEntryGetTag retrieves a copy of the tag of the specified font list entry. This routine allocates memory for the tag string that must be freed by the application.

entry	Specifies the font list entry

Return Values

Returns the tag for the font list entry. The function allocates space to hold the returned tag. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XtFree**.

Related Information

XmFontList(3), **XmFontListEntryCreate**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryLoad**(3), and **XmFontListNextEntry**(3).

XmFontListEntryLoad

Purpose

A font list function that loads a font or creates a font set and creates an accompanying font list entry

Synopsis

#include <Xm/Xm.h>

XmFontListEntry XmFontListEntryLoad(
    **Display ****display*,
    **char ****font_name*,
    XmFontType *type*,
    char ****tag*);**

Description

XmFontListEntryLoad loads a font or creates a font set based on the value of the *type* argument. It creates and returns a font list entry that contains the font or font set and the specified tag.

If the value of *type* is **XmFONT_IS_FONT**, the function uses the **XtCvtStringToFontStruct** routine to convert the value of *font_name* to a font struct. If the value of *type* is **XmFONT_IS_FONTSET**, the function uses the **XtCvtStringToFontSet** converter to create a font set in the current locale. **XmFontListEntryLoad** creates a font list entry that contains the font or font set derived from the converter. For more information about **XtCvtStringToFontStruct** and **XtCvtStringToFontSet**, see *X Toolkit Intrinsics—C Language Interface.*

display	Specifies the display where the font list will be used.
font_name	Specifies an X Logical Font Description (XLFD) string, which is interpreted either as a font name or as a base font name list. A base font name list is a comma-separated and NULL-terminated string.
type	Specifies whether the *font_name* argument refers to a font name or to a base font name list. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.

tag	Specifies the tag of the font list entry to be created. The tag may be specified as **XmFONTLIST_DEFAULT_TAG**, which is used to identify the default font list element in a font list when specified as part of a resource.

Return Values

If the specified font is not found, or if the specified font set cannot be created, then either an implementation-defined font will be opened or a font set will be created, and a warning messge will be generated. If no suitable font can be found or a font set cannot be created, then another message will be generated and the function will return NULL; otherwise the function returns a font list entry. If the function returns a font list entry, the function allocates space to hold the font list entry. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmFontListEntryFree**.

Related Information

XmFontList(3), **XmFontListAppendEntry**(3), **XmFontListEntryCreate**(3), **XmFontListEntryFree**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryGetTag**(3), and **XmFontListRemoveEntry**(3).

XmFontListFree

Purpose

A font list function that recovers memory used by a font list

Synopsis

```
#include <Xm/Xm.h>

void XmFontListFree(
        XmFontList list);
```

Description

XmFontListFree recovers memory used by a font list. This routine does not free the *XFontSet* or *XFontStruct* associated with the specified font list.

list	Specifies the font list to be freed

Related Information

XmFontList(3), **XmFontListAppendEntry**(3), **XmFontListCopy**(3), and **XmFontListRemoveEntry**(3).

XmFontListFreeFontContext

Purpose

A font list function that instructs the toolkit that the font list context is no longer needed

Synopsis

#include <Xm/Xm.h>

void XmFontListFreeFontContext(
    XmFontContext *context***);**

Description

XmFontListFreeFontContext instructs the toolkit that the context is no longer needed and will not be used without reinitialization.

context	Specifies the font list context structure that was allocated by the **XmFontListInitFontContext** function

Related Information

XmFontListInitFontContext(3) and **XmFontListNextEntry**(3).

XmFontListGetNextFont

Purpose

A font list function that allows applications to access the fonts and character sets in a font list

Synopsis

#include <Xm/Xm.h>

Boolean XmFontListGetNextFont(
    XmFontContext *context***,**
    XmStringCharSet ***charset*,**
    XFontStruct *****font*);**

Description

XmFontListGetNextFont accesses the character set and font for the next entry of the font list. The application first uses the **XmFontListInitFontContext** routine to create a font list context. The application then calls **XmFontListGetNextFont** repeatedly with the same context. Each succeeding call accesses the next element of the font list. When finished, the application calls **XmFontListFreeFontContext** to free the allocated font list context.

This routine allocates memory for the character set string that must be freed by the application. The function allocates memory for *charset*. The application is responsible for managing the allocated memory. The application can recover the allocated memory by calling **XtFree**.

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmFontListNextEntry**. If **XmFontListGetNextFont** is passed a context that contains a font set entry, it will return the first font of the font set. The next call to the function will move to the next entry in the font list.

context	Specifies the font list context
charset	Specifies a pointer to a character set string; the routine returns the character set for the current font list element

font	Specifies a pointer to a pointer to a font structure; the routine returns the font for the current font list element

Return Values

Returns True if the returned values are valid; otherwise, returns False.

Related Information

XmFontList(3) and **XmFontListNextEntry**(3).

XmFontListInitFontContext

Purpose

A font list function that allows applications to access the entries in a font list

Synopsis

#include <Xm/Xm.h>

Boolean XmFontListInitFontContext(
  XmFontContext **context*,
  XmFontList *fontlist***);**

Description

XmFontListInitFontContext establishes a context to allow applications to access the contents of a font list. This context is used when reading the font list entry tag, font, or font set associated with each entry in the font list. A Boolean status is returned to indicate whether or not the font list is valid.

If an application deallocates the font list passed to **XmFontListInitFontContext** as the fontlist argument, the context established by this function is rendered no longer valid.

context	Specifies a pointer to the allocated context
fontlist	Specifies the font list

Return Values

Returns True if the context was allocated; otherwise, returns False. If the function allocated a context, the application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmFontListFreeFontContext**.

Related Information

XmFontList(3), **XmFontListFreeFontContext**(3), and **XmFontListNextEntry**(3).

XmFontListNextEntry

Purpose

A font list function that returns the next entry in a font list

Synopsis

#include <Xm/Xm.h>

XmFontListEntry XmFontListNextEntry(
    XmFontContext *context***);**

Description

XmFontListNextEntry returns the next entry in the font list. The application uses the **XmFontListInitFontContext** routine to create a font list context. The first call to **XmFontListNextEntry** sets the context to the first entry in the font list. The application then calls **XmFontListNextEntry** repeatedly with the same context. Each succeeding call accesses the next entry of the font list. When finished, the application calls **XmFontListFreeFontContext** to free the allocated font list context.

context	Specifies the font list context

Return Values

Returns NULL if the context does not refer to a valid entry or if it is at the end of the font list; otherwise, it returns a font list entry. If the function does return a font list entry, the font list entry is not a copy. Therefore, the application should not free the returned font list entry.

Related Information

XmFontList(3), **XmFontListEntryFree**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryGetTag**(3), **XmFontListFreeFontContext**(3), and **XmFontListInitFontContext**(3).

XmFontListRemoveEntry

Purpose

A font list function that removes a font list entry from a font list

Synopsis

#include <Xm/Xm.h>

XmFontList XmFontListRemoveEntry(
    XmFontList *oldlist***,**
    XmFontListEntry *entry***);**

Description

XmFontListRemoveEntry creates a new font list that contains the contents of *oldlist* minus those entries specified in *entry*. The routine removes any entries from *oldlist* that match the components (tag, type font/font set) of the specified entry. The function deallocates the original font list after extracting the required information. The caller uses **XmFontListEntryFree** to recover memory allocated for the specified entry. This routine does not free the *XFontSet* or *XFontStruct* associated with the font list entry that is removed.

oldlist	Specifies the font list
entry	Specifies the font list entry to be removed

Return Values

If *oldlist* is NULL, the function returns NULL. If *entry* is NULL or no entries are removed, the function returns *oldlist*. Otherwise, it returns a new font list. If the function returns a new font list, the function allocates space to hold the new font list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmFontListFree**.

Related Information

XmFontList(3), **XmFontListAppendEntry**(3), **XmFontListEntryCreate**(3), **XmFontListEntryFree**(3), **XmFontListEntryLoad**(3), and **XmFontListFree**(3).

XmGetAtomName

Purpose

A function that returns the string representation for an atom

Synopsis

#include <Xm/Xm.h>
#include <Xm/AtomMgr.h>

String XmGetAtomName(
  Display * *display***,**
  Atom *atom***);**

Description

XmGetAtomName returns the string representation for an atom. It mirrors the *Xlib* interfaces for atom management but provides client-side caching. When and where caching is provided in *Xlib*, the routines will become pseudonyms for the *Xlib* routines.

display	Specifies the connection to the X server
atom	Specifies the atom for the property name you want returned

Return Values

Returns a string. The function allocates space to hold the returned string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XFree**.

XmGetColorCalculation

Purpose

A function to get the procedure used for default color calculation

Synopsis

```
#include <Xm/Xm.h>

XmColorProc XmGetColorCalculation(
        void);
```

Description

XmGetColorCalculation returns the procedure being used to calculate default colors.

For a description of **XmColorProc**, see **XmSetColorCalculation**(3).

Return Values

Returns the procedure used for default color calculation.

Related Information

XmChangeColor(3), **XmGetColors**(3), and **XmSetColorCalculation**(3).

XmGetColors

Purpose

A function that generates foreground, select, and shadow colors

Synopsis

#include <Xm/Xm.h>

void XmGetColors(
    Screen * *screen***,**
    Colormap *colormap***,**
    Pixel *background***,**
    Pixel * *foreground***,**
    Pixel * *top_shadow***,**
    Pixel * *bottom_shadow***,**
    Pixel * *select***);**

Description

XmGetColors takes a screen, a colormap, and a background pixel, and returns pixel values for foreground, select, and shadow colors.

screen Specifies the screen for which these colors should be allocated.

colormap Specifies the colormap from which these colors should be allocated.

background Specifies the background on which the colors should be based.

foreground Specifies a pointer to the returned foreground pixel value. If this argument is NULL, no value is allocated or returned for this color.

top_shadow Specifies a pointer to the returned top shadow pixel value. If this argument is NULL, no value is allocated or returned for this color.

bottom_shadow
Specifies a pointer to the returned bottom shadow pixel value. If this argument is NULL, no value is allocated or returned for this color.

select Specifies a pointer to the returned select pixel value. If this argument is NULL, no value is allocated or returned for this color.

Related Information

XmChangeColor(3), **XmGetColorCalculation**(3), and **XmSetColorCalculation**(3).

XmGetDestination

Purpose

A function that returns the widget ID of the widget to be used as the current destination for quick paste and certain clipboard operations

Synopsis

#include <Xm/Xm.h>

Widget XmGetDestination(
Display **display***);**

Description

XmGetDestination returns the widget that is the current destination on the specified display. The destination is generally the last editable widget on which a select, edit, insert, or paste operation was performed and is the destination for quick paste and certain clipboard functions. The destination is NULL until a keyboard or mouse operation has been done on an editable widget. Refer to the *Motif 2.1—Programmer's Guide* for a description of keyboard focus.

display	Specifies the display whose destination widget is to be queried

Return Values

Returns the widget ID for the current destination or NULL if there is no current destination.

XmGetDragContext

Purpose

A Drag and Drop function that retrieves the DragContext widget ID associated with a timestamp

Synopsis

#include <Xm/DragC.h>

Widget XmGetDragContext(
    Widget *refwidget*,
    Time *timestamp***);**

Description

XmGetDragContext returns the widget ID of the active DragContext associated with a given display and timestamp. A timestamp uniquely identifies which DragContext is active when more than one drag and drop transaction has been initiated on a display. If the specified timestamp matches a timestamp processed between the start and finish of a single drag and drop transaction, the function returns the corresponding DragContext ID.

refwidget	Specifies the ID of the widget that the routine uses to identify the intended display. The function returns the ID of the DragContext associated with the display value passed by this widget.
timestamp	Specifies a timestamp.

For a complete definition of DragContext and its associated resources, see **XmDragContext**(3).

Return Values

Returns the ID of the DragContext widget that is active for the specified timestamp. Otherwise, returns NULL if no active DragContext is found.

Related Information

XmDragContext(3).

XmGetFocusWidget

Purpose

Returns the ID of the widget that has keyboard focus

Synopsis

#include <Xm/Xm.h>

Widget XmGetFocusWidget(
    Widget *widget***);**

Description

XmGetFocusWidget examines the hierarchy that contains the specified widget and returns the ID of the widget that has keyboard focus. The function extracts the widget ID from the associated Shell widget; therefore, the specified widget can be located anywhere in the hierarchy.

widget	Specifies a widget ID within a given hierarchy

Return Values

Returns the ID of the widget with keyboard focus. If no child of the Shell has focus, the function returns NULL.

Related Information

XmProcessTraversal(3).

XmGetMenuCursor

Purpose

A function that returns the cursor ID for the current menu cursor

Synopsis

#include <Xm/Xm.h>

Cursor XmGetMenuCursor(
Display * *display***);**

Description

XmGetMenuCursor queries the menu cursor currently being used by this client on the specified display and returns the cursor ID. This function returns the menu cursor for the default screen of the display.

NOTE: XmGetMenuCursor is obsolete and exists for compatibility with previous releases. Instead of using this function, call **XtGetValues** for the XmScreen resource **XmNmenuCursor**.

display	Specifies the display whose menu cursor is to be queried

Return Values

Returns the cursor ID for the current menu cursor or the value None if a cursor is not yet defined. A cursor will not be defined if the application makes this call before the client has created any menus on the specified display.

Related Information

XmScreen(3).

XmGetPixmap

Purpose

A pixmap caching function that generates a pixmap, stores it in a pixmap cache, and returns the pixmap

Synopsis

```
#include <Xm/Xm.h>

Pixmap XmGetPixmap(
        Screen *screen,
        char *image_name,
        Pixel foreground,
        Pixel background);
```

Description

XmGetPixmap uses the parameter data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data. If one is found, a reference count is incremented and the pixmap is returned. Applications should use **XmDestroyPixmap** when the pixmap is no longer needed.

screen	Specifies the display screen on which the pixmap is to be drawn. The depth of the pixmap is the default depth for this screen.
image_name	Specifies the name of the image to be used to generate the pixmap
foreground	Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image
background	Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image

If a pixmap is not found, *image_name* is used to perform a lookup in the image cache. If an image is found, it is used to generate the pixmap, which is then cached and returned.

If an image is not found, the *image_name* is used as a filename, and a search is made for an **X10** or **X11** bitmap file. If it is found, the file is read, converted into an image, and cached in the image cache. The image is then used to generate a pixmap, which is cached and returned.

If *image_name* has a leading slash (/), it specifies a full pathname, and **XmGetPixmap** opens the file as specified. Otherwise, *image_name* specifies a filename. In this case, **XmGetPixmap** looks for the file along a search path specified by the **XBMLANGPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set. The default search path contains a lot of directories. Therefore, **XmGetPixmap** will need a relatively long time to search through all these directories for pixmaps and bitmaps. Applications that use a lot of pixmaps and bitmaps will probably run more quickly if **XBMLANGPATH** is set to a short list of directories. In addition to X bitmap files (XBM), Motif also supports XPM (X Pixmap) file formats. The **XBMLANGPATH** specifies the path for both XBM and XPM files. Refer to the **XmGetPixmapByDepth** reference page for further details.

The **XBMLANGPATH** environment variable specifies a search path for X bitmap files. It can contain the substitution field **%B**, where the *image_name* argument to **XmGetPixmap** is substituted for **%B**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to *bitmaps*, and **%S** is always mapped to NULL.

If **XBMLANGPATH** is not set but the environment variable **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$XAPPLRESDIR/%L/bitmaps/%N/%B**
- **$XAPPLRESDIR/%l_%t/bitmaps/%N/%B**
- **$XAPPLRESDIR/%l/bitmaps/%N/%B**
- **$XAPPLRESDIR/bitmaps/%N/%B**
- **$XAPPLRESDIR/%L/bitmaps/%B**
- **$XAPPLRESDIR/%l_%t/bitmaps/%B**
- **$XAPPLRESDIR/%l/bitmaps/%B**
- **$XAPPLRESDIR/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**

- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

If neither **XBMLANGPATH** nor **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$HOME/%L/bitmaps/%N/%B**
- **$HOME/%l_%t/bitmaps/%N/%B**
- **$HOME/%l/bitmaps/%N/%B**
- **$HOME/bitmaps/%N/%B**
- **$HOME/%L/bitmaps/%B**
- **$HOME/%l_%t/bitmaps/%B**
- **$HOME/%l/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%B**

- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

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

The following substitutions are used in these paths:

%B	The image name, from the *image_name* argument
%N	The class name of the application
%L	The display's language string. This string is influenced by **XtSetLanguageProc**. The default string is determined by calling setlocale(*LC_ALL*, *NULL*).
%l_%t	The language and territory component of the display's language string
%l	The language component of the display's language string

The contents of the file must conform to the rules for X11 bitmap files. In other words, Motif can read any X11 conformant bitmap file.

Return Values

Returns a pixmap when successful; returns **XmUNSPECIFIED_PIXMAP** if the image corresponding to *image_name* cannot be found.

Related Information

XmDestroyPixmap(3), **XmGetPixmapByDepth**(3), **XmInstallImage**(3), and **XmUninstallImage**(3).

XmGetPixmapByDepth

Purpose

A pixmap caching function that generates a pixmap, stores it in a pixmap cache, and returns the pixmap

Synopsis

#include <Xm/Xm.h>

Pixmap XmGetPixmapByDepth(
  Screen ****screen*,**
  char ****image_name*,**
  Pixel *foreground***,**
  Pixel *background***,**
  int *depth***);**

Description

XmGetPixmapByDepth uses the parameter data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data. If one is found, a reference count is incremented and the pixmap is returned. Applications should use **XmDestroyPixmap** when the pixmap is no longer needed.

screen	Specifies the display screen on which the pixmap is to be drawn
image_name	Specifies the name of the image to be used to generate the pixmap
foreground	Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image
background	Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image
depth	Specifies the depth of the pixmap

If a matching pixmap is not found, *image_name* is used to perform a lookup in the image cache. If an image is found, it is used to generate the pixmap, which is then cached and returned.

If an image is not found, *image_name* is used as a filename, and a search is made for an **X10** or **X11** bitmap file. If it is found, the file is read, converted into an image, and cached in the image cache. The image is then used to generate a pixmap, which is cached and returned.

If *image_name* has a leading / (slash), it specifies a full pathname, and **XmGetPixmapByDepth** opens the file as specified. Otherwise, *image_name* specifies a filename. In this case, **XmGetPixmapByDepth** looks for the file along a search path specified by the **XBMLANGPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set. The default search path contains a lot of directories. Therefore, **XmGetPixmapByDepth** will need a relatively long time to search through all these directories for pixmaps and bitmaps. Applications that use a lot of pixmaps and bitmaps will probably run more quickly if **XBMLANGPATH** is set to a short list of directories. In addition to X bitmap files (XBM), Motif also supports XPM (X Pixmap) file formats. The **XBMLANGPATH** specifies the path for both XBM and XPM files. XPM files are described in more detail later in this reference page.

The **XBMLANGPATH** environment variable specifies a search path for X bitmap files. It can contain the substitution field **%B**, where the *image_name* argument to **XmGetPixmapByDepth** is substituted for **%B**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to *bitmaps*, and **%S** is always mapped to NULL.

If **XBMLANGPATH** is not set, but the environment variable **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$XAPPLRESDIR/%L/bitmaps/%N/%B**
- **$XAPPLRESDIR/%l_%t/bitmaps/%N/%B**
- **$XAPPLRESDIR/%l/bitmaps/%N/%B**
- **$XAPPLRESDIR/bitmaps/%N/%B**
- **$XAPPLRESDIR/%L/bitmaps/%B**
- **$XAPPLRESDIR/%l_%t/bitmaps/%B**
- **$XAPPLRESDIR/%l/bitmaps/%B**
- **$XAPPLRESDIR/bitmaps/%B**
- **$HOME/bitmaps/%B**

- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

If neither **XBMLANGPATH** nor **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$HOME/%L/bitmaps/%N/%B**
- **$HOME/%l_%t/bitmaps/%N/%B**
- **$HOME/%l/bitmaps/%N/%B**
- **$HOME/bitmaps/%N/%B**
- **$HOME/%L/bitmaps/%B**
- **$HOME/%l_%t/bitmaps/%B**
- **$HOME/%l/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l_%t/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**

- **/usr/lib/X11/%l_%t/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

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

The following substitutions are used in these paths:

%B	The image name, from the *image_name* argument
%N	The class name of the application
%L	The display's language string. This string is influenced by **XtSetLanguageProc**. The default string is determined by calling setlocale(*LC_ALL*, *NULL*).
%l_%t	The language and territory component of the display's language string
%l	The language component of the display's language string

The contents of the file must conform to the rules for X11 bitmap files. In other words, Motif can read any X11 conformant bitmap file.

The XPM file format is used for storing or getting back colored X pixmaps from files. The XPM library is provided as unsupported with Motif. To build applications without XPM, use the *NO_XPM* macro. The following shows both XBM and XPM files, respectively, for a plaid pattern.

```
/* XBM file */
#define plaid_width 22
#define plaid_height 22
#define plaid_x_hot -1
#define plaid_y_hot -1
static char plaid_bits[] = {
   0x75, 0xfd, 0x3f, 0xaa, 0xfa, 0x3e, 0x75, 0xfd, 0x3f, 0xaa, 0xfa, 0x3e,
   0x75, 0xfd, 0x3f, 0xff, 0x57, 0x15, 0x75, 0xfd, 0x3f, 0xaa, 0xfa, 0x3e,
   0x75, 0xfd, 0x3f, 0xaa, 0xfa, 0x3e, 0x75, 0xfd, 0x3f, 0x20, 0xa8, 0x2b,
   0x20, 0x50, 0x15, 0x20, 0xa8, 0x2b, 0x20, 0x50, 0x15, 0x20, 0xa8, 0x2b,
   0xff, 0xff, 0x3f, 0x20, 0xa8, 0x2b, 0x20, 0x50, 0x15, 0x20, 0xa8, 0x2b,
   0x20, 0x50, 0x15, 0x20, 0xa8, 0x2b};
```

```
/* XPM file */
static char * plaid[] = {
/* plaid pixmap
 * width height ncolors chars_per_pixel */
"22 22 4 2 ",
/* colors */
"   c red       m white  s light_color ",
"Y  c green     m black  s lines_in_mix ",
"+  c yellow    m white  s lines_in_dark ",
"x              m black  s dark_color ",
/* pixels */
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"Y Y Y Y Y x Y Y Y Y Y + x + x + x + x + x + ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"x x x x x x x x x x x x x x x x x x x x x x ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x "
};
```

Return Values

Returns a pixmap when successful; returns **XmUNSPECIFIED_PIXMAP** if the image corresponding to *image_name* cannot be found.

Related Information

XmDestroyPixmap(3), **XmInstallImage**(3), and **XmUninstallImage**(3).

XmGetPostedFromWidget

Purpose

A RowColumn function that returns the widget from which a menu was posted

Synopsis

#include <Xm/RowColumn.h>

Widget XmGetPostedFromWidget(
Widget *menu***);**

Description

XmGetPostedFromWidget returns the widget from which a menu was posted. For torn-off menus, this function returns the widget from which the menu was originally torn. An application can use this routine during the activate callback to determine the context in which the menu callback should be interpreted.

menu	Specifies the widget ID of the menu

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the widget ID of the widget from which the menu was posted. If the menu is a Popup Menu, the returned widget is the widget from which the menu was popped up. If the menu is a Pulldown Menu, the returned widget is the MenuBar or OptionMenu from which the widget was pulled down.

Related Information

XmRowColumn(3).

XmGetSecondaryResourceData

Purpose

A function that provides access to secondary widget resource data

Synopsis

#include <Xm/Xm.h>

Cardinal XmGetSecondaryResourceData(
        WidgetClass *widget_class***,**
        XmSecondaryResourceData *****secondary_data_return*);**

Description

Some Motif widget classes (such as Gadget, Text, and VendorShell) have resources that are not accessible through the functions **XtGetResourceList** and **XtGetConstraintResourceList**. In order to retrieve the descriptions of these resources, an application must use **XmGetSecondaryResourceData**.

When a widget class has such resources, this function provides descriptions of the resources in one or more data structures. **XmGetSecondaryResourceData** takes a widget class argument and returns the number of these data structures associated with the widget class. If the return value is greater than 0 (zero), the function allocates and fills an array of pointers to the corresponding data structures. It returns this array at the address that is the value of the *secondary_data_return* argument.

The type **XmSecondaryResourceData** is a pointer to a structure with two members that are useful to an application: *resources*, of type *XtResourceList*, and **num_resources**, of type **Cardinal**. The *resources* member is a list of the widget resources that are not accessible using Xt functions. The **num_resources** member is the length of the *resources* list.

If the return value is greater than 0 (zero), **XmGetSecondaryResourceData** allocates memory that the application must free. Use **XtFree** to free the resource list in each structure (the value of the *resources* member), the structures themselves, and the array of pointers to the structures (the array whose address is *secondary_data_return*).

widget_class	Specifies the widget class for which secondary resource data is to be retrieved.

secondary_data_return
Specifies a pointer to an array of **XmSecondaryResourceData** pointers to be returned by this function. If the widget class has no secondary resource data, for example, if the value returned by the function is 0 (zero), the function returns no meaningful value for this argument.

Return Values

Returns the number of secondary resource data structures associated with this widget class.

Examples

The following example uses **XmGetSecondaryResourceData** to print the names of the secondary resources of the Motif Text widget and then frees the data allocated by the function:

```
XmSecondaryResourceData * block_array;
Cardinal num_blocks, i, j;
if (num_blocks = XmGetSecondaryResourceData (xmTextWidgetClass,
                                             &block_array)) {
  for (i = 0; i < num_blocks; i++) {
    for (j = 0; j < block_array[i]->num_resources; j++) {
      printf("%s\n", block_array[i]->resources[j].resource_name);
    }
    XtFree((char*)block_array[i]->resources);
    XtFree((char*)block_array[i]);
  }
  XtFree((char*)block_array);
}
```

XmGetTabGroup

Purpose

Returns the widget ID of a tab group

Synopsis

```
#include <Xm/Xm.h>

Widget XmGetTabGroup(
        Widget widget);
```

Description

XmGetTabGroup returns the widget ID of the tab group that contains the specified widget.

widget	Specifies a widget ID within a tab group

Return Values

Returns the widget ID of a tab group or shell, determined as follows:

- If *widget* is a tab group or shell, returns *widget*
- If neither *widget* nor any ancestor up to the nearest shell is a tab group, returns the nearest ancestor of *widget* that is a shell
- Otherwise, returns the nearest ancestor of *widget* that is a tab group

Related Information

XmAddTabGroup(3), **XmManager**(3), and **XmPrimitive**(3).

XmGetTearOffControl

Purpose

A RowColumn function that obtains the widget ID for the tear-off control in a menu

Synopsis

#include <Xm/RowColumn.h>

Widget XmGetTearOffControl(
  Widget *menu***);**

Description

XmGetTearOffControl provides the application with the means for obtaining the widget ID of the internally created tear-off control in a tear-off menu.

RowColumn creates a tear-off control for a PulldownMenu or PopupMenu when the **XmNtearOffModel** resource is initialized or set to **XmTEAR_OFF_ENABLED**. The tear-off control is a widget that appears as the first element in the menu. The user tears off the menu by means of mouse or keyboard events in the tear-off control.

The tear-off control has Separator-like behavior. Once the application has obtained the widget ID of the tear-off control, it can set resources to specify the appearance of the control. The application or user can also set these resources in a resource file by using the name of the control, which is **TearOffControl**. For a list of the resources the application or user can set, see **XmRowColumn**(3).

menu	Specifies the widget ID of the RowColumn PulldownMenu or PopupMenu

For more information on tear-off menus and a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the widget ID for the tear-off control, or NULL if no tear-off control exists. An application should not assume that the returned widget will be of any particular class.

Related Information

XmRowColumn(3).

XmGetVisibility

Purpose

A function that determines if a widget is visible

Synopsis

```
#include <Xm/Xm.h>

XmVisibility XmGetVisibility(
        Widget widget);
```

Description

XmGetVisibility returns the visibility state of the specified widget. It checks to see if some part of the widget's rectangular area is unobscured by the widget's ancestors, or some part of the widget's rectangular area is inside the work window (but possibly outside the clip window) of a ScrolledWindow whose **XmNscrollingPolicy** is **XmAUTOMATIC** and whose **XmNtraverseObscuredCallback** is not NULL.

XmGetVisibility does not check to see if *widget* is obscured by its siblings or by siblings of its ancestors. Consequently, **XmGetVisibility** returns **XmVISIBILITY_UNOBSCURED** for widgets which are completely or partially covered by one or more siblings of *widget* by one or more siblings of ancestors of *widget*.

When a widget which is unrealized is being queried, it is indicated that the widget is fully obscured. If an application unmaps a *widget* that has its **XmNmappedWhenManaged** resource set to True, the return value is undefined. When a widget which is unmanaged is being queried, it is indicated that the widget is fully obscured.

widget	Specifies the ID of the widget

Return Values

Returns one of the following values:

XmVISIBILITY_UNOBSCURED
Indicates that the widget is mapped, not obscured, and is completely visible on the screen.

XmVISIBILITY_PARTIALLY_OBSCURED
Indicates that the widget is mapped, and is not completely visible on the screen (partially obscured).

XmVISIBILITY_FULLY_OBSCURED
Indicates that the widget is not at all visible on the screen.

Related Information

XmIsTraversable(3), **XmManager**(3), and **XmProcessTraversal**(3).

XmGetXmDisplay

Purpose

A Display function that returns the XmDisplay object ID for a specified display

Synopsis

```
#include <Xm/Display.h>

Widget XmGetXmDisplay(
        Display *display);
```

Description

XmGetXmDisplay returns the **XmDisplay** object ID associated with a display. The application can access Display resources with **XtGetValues**.

display	Specifies the display for which the **XmDisplay** object ID is to be returned

For a complete definition of Display and its associated resources, see **XmDisplay**(3).

Return Value

Returns the **XmDisplay** object ID for the specified display.

Related Information

XmDisplay(3).

XmGetXmScreen

Purpose

A Screen function that returns the XmScreen object ID for a specified screen

Synopsis

```
#include <Xm/Screen.h>

Widget XmGetXmScreen(
        Screen *screen);
```

Description

XmGetXmScreen returns the **XmScreen** object ID associated with a screen. The application can access and manipulate Screen resources with **XtGetValues** and **XtSetValues**.

screen	Specifies the screen for which the **XmScreen** ID is to be returned

For a complete definition of Screen and its associated resources, see **XmScreen**(3).

Return Values

Returns the **XmScreen** object ID.

Related Information

XmScreen(3).

XmImCloseXIM

Purpose

An input manager function that releases the input method associated with a specified widget

Synopsis

#include <Xm/XmIm.h>

void XmImCloseXIM(
        Widget *widget***);**

Description

XmImCloseXIM closes all input contexts associated with the Input Method (IM) of *widget*. *widget* is used to identify the Display that specifies the Input Method opened for the widget. Upon closure, all widgets registered with the input contexts are unregistered. Also, the Input Method specified by Display is closed.

widget	Specifies the ID of a widget whose reference Input Method is to be closed.

Related Information

XmImGetXIM(3) and **XmImRegister**(3).

XmImFreeXIC

Purpose

An input manager function that unregisters widgets for an XIC

Synopsis

#include <Xm/XmIm.h>

void XmImFreeXIC(
    Widget *widget***,**
    XIC *xic***);**

Description

XmImFreeXIC unregisters all widgets associated with the specified X Input Context (XIC). The specified *widget* must be associated with the specified *xic*.

After unregistering the associated widgets, this call frees the *xic*.

widget	Specifies the ID of a widget used to identify the **VendorShell** and **XmDisplay** that maintain the widget-XIC registry.
xic	Specifies the Input Context associated with the widget.

Related Information

XmImGetXIC(3) and **XmImSetXIC**(3).

XmImGetXIC

Purpose

An input manager function that obtains an XIC for a widget

Synopsis

#include <Xm/XmIm.h>

XIC XmImGetXIC(
    Widget *widget***,**
    XmInputPolicy *input_policy***,**
    ArgList *args***,**
    Cardinal *num_args***);**

Description

XmImGetXIC creates and registers an X Input Context (XIC) with the specified arguments for *widget*. If **XmINHERIT_POLICY** is specified for *input_policy*, a new XIC will be created only if required to by the arguments or by the **VendorShell** input policy. Any existing XIC registered with *widget* is unregistered.

Refer to the **VendorShell** reference page for further details.

widget Specifies the ID of a widget for which an Input Context is to be created.

input_policy Specifies the type of input policy. It accepts the following values:

XmINHERIT_POLICY
Inherits the policy from **VendorShell**.

XmPER_WIDGET
Creates a new XIC for this widget.

XmPER_SHELL
Creates a new XIC for the shell, if needed.

args Specifies an *XtArgList* parameter to use for creating the XIC.

num_args Specifies the number of arguments in the *args* parameter.

Return Values

Returns the created XIC. The application is responsible for freeing the returned XIC by calling **XmImFreeXIC**.

Related Information

XmImSetXIC(3) and **XmImFreeXIC**(3).

XmImGetXIM

Purpose

An input manager function that retrieves the input method associated with a specified widget

Synopsis

#include <Xm/XmIm.h>

XIM XmImGetXIM(
  Widget *widget***);**

Description

XmImGetXIM retrieves the XIM data structure representing the input method that the input manager has opened for the specified widget. If an input method has not been opened by a previous call to **XmImRegister**, the first time this routine is called it opens an input method using the **XmNinputMethod** resource for the VendorShell. If the **XmNinputMethod** is NULL, an input method is opened using the current locale. If it cannot open an input method, the function returns NULL.

widget	Specifies the ID of a widget registered with the input manager

Return Values

Returns the input method for the current locale associated with the specified widget's input manager; otherwise, returns NULL. The application is responsible for freeing the returned XIM by calling **XmImCloseXIM**.

Related Information

XmImCloseXIM(3), **XmImGetXIM**(3), **XmImMbLookupString**(3), and **XmImRegister**(3).

XmImMbLookupString

Purpose

An input manager function that retrieves a composed string from an input method

Synopsis

```
#include <Xm/XmIm.h>

int XmImMbLookupString(
        Widget widget,
        XKeyPressedEvent *event,
        char *buffer_return,
        int bytes_buffer,
        KeySym *keysym_return,
        int *status_return);
```

Description

XmImMbLookupString returns a string composed in the locale associated with the widget's input method and a KeySym that is currently mapped to the keycode in a KeyPress event. The KeySym is obtained by using the standard interpretation of Shift, Lock and Group modifiers as defined in the X Protocol specification.

An XIM will be created, but an XIC will not be created. One of the functions, **XmImSetValues**, **XmImVaSetValues**, or **XmImGetXIC**, needs to be called to create an XIC.

widget Specifies the ID of the widget registered with the input manager

event Specifies the key press event

buffer_return
Specifies the buffer in which the string is returned

bytes_buffer Specifies the size of the buffer in bytes

keysym_return
Specifies a pointer to the KeySym returned if one exists

status_return

Specifies the status values returned by the function. These status values are the same as those for the **XmbLookupString** function. The possible status values are:

XBufferOverflow

The size of the buffer was insufficient to handle the returned string. The contents of *buffer_return* and *keysym_return* are not modified. The required buffer size is returned as a value of the function. The client should repeat the call with a larger buffer size to receive the string.

XLookupNone

No consistent input was composed. The contents of *buffer_return* and *keysym_return* are not modified and the function returns a value of 0.

XLookupChars

Some input characters were composed and returned in *buffer_return*. The content of *keysym_return* is not modified. The function returns the length of the string in bytes.

XLookupKeysym

A keysym value was returned instead of a string. The content of *buffer_return* is not modified and the function returns a value of 0.

XLookupBoth

A keysym value and a string were returned. The keysym value may not necessarily correspond to the string returned. The function returns the length of the string in bytes.

Return Values

Return values depend on the status returned by the function. Refer to the description of status values above.

Related Information

XmImGetXIM(3), **XmImGetXIC**(3), **XmImRegister**(3), **XmImSetValues**(3), and **XmImUnregister**(3).

XmImMbResetIC

Purpose

An input manager function that resets the input context for a widget

Synopsis

#include <Xm/XmIm.h>

void XmImMbResetIC(
        Widget *widget***,**
        char ***mb***);**

Description

XmImMbResetIC gets the XIC of the widget and resets it. It puts a pointer to a string containing the current preedit string to *mb*. The caller should free the returned string after use by calling **Xfree**.

widget	Specifies the ID of the widget.
mb	Contains a pointer to the preedit string upon return.

Return Values

None

Related Information

XmImRegister

Purpose

An input manager function that registers a widget with an input manager

Synopsis

#include <Xm/XmIm.h>

void XmImRegister(
  Widget *widget***,**
  unsigned int *reserved***);**

Description

XmImRegister registers a widget with its input manager. This adds the specified widget to a list of widgets that are supported by the input manager for an input method. If an input method has not been opened by a previous call to **XmImRegister**, the first time this routine is called it opens an input method using the **XmNinputMethod** resource for the VendorShell. If the **XmNinputMethod** is NULL, an input method is opened using the current locale.

If an input method cannot be opened in the current locale, **XLookupString** provides input processing.

The application is responsible for unregistering a widget by calling **XmImUnregister**.

Note that the Text, TextField, and List widgets already call the **XmImRegister** function internally. You should not call this function for these widgets before calling **XmImUnregister** first.

widget	Specifies the ID of the widget to be registered.
reserved	This argument is not used in the current release of Motif. The value should always be 0 (zero).

Related Information

XmImGetXIM(3), **XmImMbLookupString**(3), and **XmImUnregister**(3).

XmImSetFocusValues

Purpose

An input manager function that notifies an input manager that a widget has received input focus and updates the input context attributes

Synopsis

#include <Xm/XmIm.h>

void XmImSetFocusValues(
  Widget *widget***,**
  ArgList *arglist***,**
  Cardinal *argcount***,**
  );

Description

XmImSetFocusValues notifies the input manager that the specified widget has received input focus. This function also updates the attributes of the input context associated with the widget. The focus window for the XIC is set to the window of the widget. The *arglist* argument is a list of attribute/value pairs for the input context. This function passes the attributes and values to *XICSetValues*. The caller of this routine should pass in only those values that have changed since the last call to any of these functions; **XmImSetValues**, **XmImSetFocusValues**, **XmImVaSetValues**, or **XmImVaSetFocusValues**. See the description in the **XmImSetValues**(3) reference page for a list of associated resources.

If the previous parameters for the widget's XIC do not allow the previously registered XIC to be reused, that XIC will be unregistered, and a new one will be created and registered with the widget. Note that sharing of data is preserved.

widget Specifies the ID of the widget registered with the input manager.

arglist Specifies the list of attribute/value pairs to be passed to *XICSetValues*. See the description in the **XmImSetValues**(3) man page for a description of resources.

argcount Specifies the number of attribute/values pairs in the argument list (*arglist)*

Note that the Text and TextField widgets call the **XmImSetFocusValues** function when they receive focus. Therefore, further calls to the **XmImSetFocusValues** function for these widgets are unnecessary.

Related Information

XmImSetValues(3), **XmImVaSetFocusValues**(3), and **XmImVaSetValues**(3).

XmImSetValues

Purpose

An input manager function that updates attributes of an input context

Synopsis

#include <Xm/XmIm.h>

```
void XmImSetValues(
        Widget widget,
        ArgList arglist,
        Cardinal argcount,
        );
```

Description

XmImSetValues updates attributes of the input context associated with the specified widget. The *arglist* argument is a list of attribute/value pairs for the input context. This function passes the attributes and values to *XICSetValues*. The initial call to this routine should pass in all of the input context attributes. Thereafter, the application programmer calls **XmImSetValues**, for an XIC, only if a value has changed.

If the previous parameters for the widget's XIC do not allow the previously registered XIC to be reused, that XIC will be unregistered, and a new one will be created and registered with the widget. Note that sharing of data is preserved.

Note that the Text and TextField widgets call the **XmImSetValues** function when they receive focus. Therefore, further calls to the **XmImSetValues** function for these widgets are unnecessary.

widget	Specifies the ID of the widget registered with the input manager
arglist	Specifies the list of attribute/value pairs to be passed to *XICSetValues*; the following attributes are accepted: *XmNpreeditStartCallback* *XmNpreeditDoneCallback* *XmNpreeditDrawCallback* and *XmNpreeditCaretCallback*. These attributes accept an accompanying value of type pointer to structure of type *XIMCallback*.

These callbacks are used only when the *XmNpreeditType* resource of the relevant *VendorShell* has the "onthespot" value, and that the XIM supports *XIMPreeditCallbacks* input style. These values are ignored if the condition is not met.

For each of these callbacks, if the callback value is not set by this function, no action will be taken when the Input Method tries to call this callback. Refer to the "Xlib - C Language X Interface, X Version 11, Release 6," Chapter 13 for the detail of these callbacks.

argcount Specifies the number of attribute/values pairs in the argument list (*arglist)*

Resources that can be set for the input context include:

XmNbackground
Specifies the pixel value for the background color.

XmNbackgroundPixmap
Specifies a pixmap for tiling the background.

XmNfontList
Specifies the font list used by the widget. The input method uses the first occurrence of a font set tagged with **XmFONTLIST_DEFAULT_TAG**. If no such instance is found, the first font set in the font list is used. If the font list does not contain a font set, a value is not passed to *XICSetValues*.

XmNforeground
Specifies the pixel value for the foreground color.

XmNlineSpace
Specifies the line spacing used in the pre-edit window.

XmNrenderTable
Specifies the render table used by the widget.

XmNspotLocation
Specifies the *x* and *y* coordinates of the position where text will be inserted in the widget handling input, whose input method style is **"OverTheSpot"**. The *y* coordinate is the position of the baseline used by the current text line.

The caller may also pass any other vendor-defined resources to this function. For additional information on the internationalization interface, see the Xlib documentation.

Related Information

XmImSetFocusValues(3), **XmImVaSetFocusValues**(3), and **XmImVaSetValues**(3).

XmImSetXIC

Purpose

An input manager function that registers an existing XIC with a widget

Synopsis

#include <Xm/XmIm.h>

XIC XmImSetXIC(
    Widget *widget***,**
    XIC *xic***);**

Description

XmImSetXIC registers the specified X Input Context (XIC) with *widget*. Any existing XIC registered for *widget* is unregistered. The new XIC registered for *widget* is returned.

If *xic* was not created by **XmImGetXIC** or **XmImRegister**, it will not be subject to closing activities when it has no widgets registered with it.

widget	Specifies the ID of a widget for which a new Input Context is to be registered.
xic	Specifies the Input Context to be registered with the widget. If *xic* is NULL, the function returns the current *XIC* used by *widget*.

Return Values

Returns the new XIC registered for *widget*. The application is responsible for freeing the returned XIC. To free the XIC, call **XmImFreeXIC**.

Related Information

XmImGetXIC(3) and **XmImFreeXIC**(3).

XmImUnregister

Purpose

An input manager function that removes a widget from association with its input manager

Synopsis

#include <Xm/XmIm.h>

void XmImUnregister(
        Widget *widget***);**

Description

XmImUnregister removes the specified widget from the list of widgets registered for input by the input manager.

Note that the Text, TextField, and List widgets already call the **XmImRegister** internally. You should call the **XmImUnregister** function for these widgets before calling **XmImRegister**.

widget	Specifies the ID of the widget to be unregistered

Related Information

XmImRegister(3).

XmImUnsetFocus

Purpose

An input manager function that notifies an input method that a widget has lost input focus

Synopsis

#include <Xm/XmIm.h>

void XmImUnsetFocus(
        Widget *widget***);**

Description

XmImUnsetFocus unsets a specified widget's focus, then notifies the input manager that the specified widget has lost its input focus.

Note that the Text, TextField, and List widgets already call the **XmImUnsetFocus** internally. Therefore, further calls to the **XmImUnsetFocus** function for those widgets are unnecessary.

widget	Specifies the ID of the widget registered with the input manager

Related Information

XmImSetFocusValues(3) and **XmImVaSetFocusValues**(3).

XmImVaSetFocusValues

Purpose

An input manager function that notifies an input manager that a widget has received input focus and updates the input context attributes

Synopsis

#include <Xm/XmIm.h>

void XmImVaSetFocusValues(
        Widget *widget***);**

Description

XmImVaSetFocusValues notifies the input manager that the specified widget has received input focus. This function also updates the attributes of the input context associated with the widget. This function passes the attributes and values to *XICSetValues*. The caller of this routine should pass in only those values that have changed since the last call to any of these functions; **XmImVaSetValues**, **XmImVaSetFocusValues**, **XmImSetValues**, or **XmImSetFocusValues**. See the description in the **XmImSetValues**(3) reference page for a list of associated resources.

This routine uses the ANSI C variable-length argument list (varargs) calling conventions. The variable-length argument list consists of groups of arguments each of which contains an attribute followed by the value of the attribute. The last argument in the list must be NULL

Note that the List and TextField widgets call the **XmImVaSetFocusValues** function when they receive focus. Therefore, further calls to the **XmImVaSetFocusValues** function for these widgets are unnecessary.

widget	Specifies the ID of the widget registered with the input manager

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

Related Information

XmImSetFocusValues(3), **XmImSetValues**(3), and **XmImVaSetValues**(3).

XmImVaSetValues

Purpose

An input manager function that updates attributes of an input context

Synopsis

```
#include <Xm/XmIm.h>

void XmImVaSetValues(
        Widget widget);
```

Description

XmImVaSetValues updates attributes of the input context associated with the specified widget. This function passes the attributes to *XICSetValues*. The initial call to this routine should pass in all of the input context attributes. Thereafter, the application programmer calls **XmImVaSetValues** only if a value has changed. See the description in the **XmImSetValues**(3) man page for a list of associated resources.

This routine uses the ANSI C variable-length argument list (varargs) calling convention. The variable-length argument list consists of groups of arguments each of which contains an attribute followed by the value of the attribute. The last argument in the list must be NULL.

Note that the List and TextField widgets call the **XmImVaSetValues** function internally. Therefore, further calls to the **XmImVaSetValues** function for these widgets are unnecessary.

widget Specifies the ID of the widget registered with the input manager

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

Related Information

XmImSetFocusValues(3), **XmImSetValues**(3), and **XmImVaSetFocusValues**(3).

XmInstallImage

Purpose

A pixmap caching function that adds an image to the image cache

Synopsis

```
#include <Xm/Xm.h>

Boolean XmInstallImage(
        XImage * image,
        char * image_name);
```

Description

XmInstallImage stores an image in an image cache that can later be used to generate a pixmap. Part of the installation process is to extend the resource converter used to reference these images. The resource converter is given the image name so that the image can be referenced in a **.Xdefaults** file. Since an image can be referenced by a widget through its pixmap resources, it is up to the application to ensure that the image is installed before the widget is created.

image Points to the image structure to be installed. The installation process does not make a local copy of the image. Therefore, the application should not destroy the image until it is uninstalled from the caching functions.

image_name Specifies a string that the application uses to name the image. After installation, this name can be used in **.Xdefaults** for referencing the image. A local copy of the name is created by the image caching functions.

The image caching functions provide a set of eight preinstalled images. These names can be used within a **.Xdefaults** file for generating pixmaps for the resource for which they are provided.

Image Name	**Description**
background	A tile of solid background
25_foreground	A tile of 25% foreground, 75% background
50_foreground	A tile of 50% foreground, 50% background
75_foreground	A tile of 75% foreground, 25% background
horizontal	A tile of horizontal lines of the two colors
vertical	A tile of vertical lines of the two colors
slant_right	A tile of slanting lines of the two colors
slant_left	A tile of slanting lines of the two colors
menu_cascade	A tile of an arrow of the foreground color
menu_checkmark	A tile of a checkmark of the foreground color
menu_dash	A tile of one horizontal line of the foreground color

Return Values

Returns True when successful; returns False if NULL *image*, NULL *image_name*, or duplicate *image_name* is used as a parameter value.

Related Information

XmUninstallImage(3), **XmGetPixmap**(3), and **XmDestroyPixmap**(3).

XmInternAtom

Purpose

A macro that returns an atom for a given name

Synopsis

#include <Xm/AtomMgr.h>

Atom XmInternAtom(
    Display * *display***,**
    String *name***,**
    Boolean *only_if_exists***);**

Description

XmInternAtom returns an atom for a given name. The returned atom remains defined even after the client's connection closes. The returned atom becomes undefined when the last connection to the X server closes.

display Specifies the connection to the X server

name Specifies the name associated with the atom you want returned. The value of *name* is case dependent.

only_if_exists
Specifies a Boolean value. If False, the atom is created even if it does not exist. (If it does not exist, the returned atom will be **None**.) If True, the atom is created only if it exists.

Return Values

Returns an atom.

XmIsMotifWMRunning

Purpose

A function that determines whether the window manager is running

Synopsis

#include <Xm/Xm.h>

Boolean XmIsMotifWMRunning(
    Widget *shell***);**

Description

XmIsMotifWMRunning lets a user know whether the Motif Window Manager is running on a screen that contains a specific widget hierarchy. This function first sees whether the _MOTIF_WM_INFO property is present on the root window of the shell's screen. If it is, its window field is used to query for the presence of the specified window as a child of root.

shell Specifies the shell whose screen will be tested for **mwm**'s presence.

Return Values

Returns True if MWM is running.

XmIsTraversable

Purpose

A function that identifies whether a widget can be traversed

Synopsis

#include <Xm/Xm.h>

Boolean XmIsTraversable(
    Widget *widget***);**

Description

XmIsTraversable determines whether the specified widget is eligible to receive focus through keyboard traversal. In general, a widget is eligible to receive focus when all of the following conditions are true:

- The widget and its ancestors are not being destroyed, are sensitive, and have a value of True for **XmNtraversalOn**.
- The widget and its ancestors are realized, managed, and (except for gadgets) mapped. If an application unmaps a *widget* that has its **XmNmappedWhenManaged** resource set to True, the return value is undefined.
- Some part of the widget's rectangular area is unobscured by the widget's ancestors, or some part of the widget's rectangular area is inside the work window (but possibly outside the clip window) of a ScrolledWindow whose **XmNscrollingPolicy** is **XmAUTOMATIC** and whose **XmNtraverseObscuredCallback** is not NULL.

Some widgets may not be eligible to receive focus even if they meet all these conditions. For example, most managers cannot receive focus through keyboard traversal. Some widgets may be eligible to receive focus under particular conditions. For example, a DrawingArea is eligible to receive focus if it meets the conditions above and has no child whose **XmNtraversalOn** resource is True.

Note that when all widgets in a shell hierarchy have been made untraversable, they are considered to have lost focus. When a widget in this hierarchy is made traversable again, it regains focus.

XmIsTraversable may return unexpected results when *widget* or its ancestors are overlapped by their siblings.

widget	Specifies the ID of the widget

Return Values

Returns True if the widget is eligible to receive focus through keyboard traversal; otherwise, returns False.

Related Information

XmGetVisibility(3) and **XmProcessTraversal**(3).

XmListAddItem

Purpose

A List function that adds an item to the list

Synopsis

#include <Xm/List.h>

void XmListAddItem(
  Widget *widget***,**
  XmString *item***,**
  int *position***);**

Description

XmListAddItem adds an item to the list at the given position. When the item is inserted into the list, it is compared with the current **XmNselectedItems** list. If the new item matches an item on the selected list, it appears selected.

widget	Specifies the ID of the List to which an item is added.
item	Specifies the item to be added to the list.
position	Specifies the position of the new item in the list. A value of 1 makes the new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the new item the last item in the list.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListAddItemUnselected

Purpose

A List function that adds an item to the list

Synopsis

#include <Xm/List.h>

void XmListAddItemUnselected(
    Widget *widget***,**
    XmString *item***,**
    int *position***);**

Description

XmListAddItemUnselected adds an item to the list at the given position. The item does not appear selected, even if it matches an item in the current **XmNselectedItems** list.

widget	Specifies the ID of the List from whose list an item is added.
item	Specifies the item to be added to the list.
position	Specifies the position of the new item in the list. A value of 1 makes the new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the new item the last item in the list.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListAddItems

Purpose

A List function that adds items to the list

Synopsis

#include <Xm/List.h>

void XmListAddItems(
    Widget *widget*,
    XmString * *items*,
    int *item_count*,
    int *position***);**

Description

XmListAddItems adds the specified items to the list at the given position. The first *item_count* items of the *items* array are added to the list. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget	Specifies the ID of the List to which an item is added.
items	Specifies a pointer to the items to be added to the list.
item_count	Specifies the number of items in *items*. This number must be nonnegative.
position	Specifies the position of the first new item in the list. A value of 1 makes the first new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the first new item follow the last item in the list.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListAddItemsUnselected

Purpose

A List function that adds items to a list

Synopsis

#include <Xm/List.h>

void XmListAddItemsUnselected(
Widget *widget***,**
XmString ****items*,**
int *item_count***,**
int *position***);**

Description

XmListAddItemsUnselected adds the specified items to the list at the given position. The inserted items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget	Specifies the ID of the List widget to add items to.
items	Specifies a pointer to the items to be added to the list.
item_count	Specifies the number of elements in *items*. This number must be nonnegative.
position	Specifies the position of the first new item in the list. A value of 1 makes the first new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the first new item follow the last item of the list.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeleteAllItems

Purpose

A List function that deletes all items from the list

Synopsis

```
#include <Xm/List.h>

void XmListDeleteAllItems(
        Widget widget);
```

Description

XmListDeleteAllItems deletes all items from the list.

widget Specifies the ID of the List from whose list the items are deleted

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeleteItem

Purpose

A List function that deletes an item from the list

Synopsis

#include <Xm/List.h>

void XmListDeleteItem(
    Widget *widget***,**
    XmString *item***);**

Description

XmListDeleteItem deletes the first item in the list that matches *item*. A warning message appears if the item does not exist.

widget	Specifies the ID of the List from whose list an item is deleted.
item	Specifies the text of the item to be deleted from the list. If *item* appears more than once in the List, only the first occurrence is matched.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeleteItems

Purpose

A List function that deletes items from the list

Synopsis

```
#include <Xm/List.h>

void XmListDeleteItems(
        Widget widget,
        XmString *items,
        int item_count);
```

Description

XmListDeleteItems deletes the specified items from the list. For each element of *items*, the first item in the list that matches that element is deleted. A warning message appears if any of the items do not exist.

widget	Specifies the ID of the List from whose list an item is deleted
items	Specifies a pointer to items to be deleted from the list
item_count	Specifies the number of elements in *items* This number must be nonnegative.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeleteItemsPos

Purpose

A List function that deletes items from the list starting at the given position

Synopsis

#include <Xm/List.h>

void XmListDeleteItemsPos(
 Widget *widget***,**
 int *item_count***,**
 int *position***);**

Description

XmListDeleteItemsPos deletes the specified number of items from the list starting at the specified position.

widget	Specifies the ID of the List from whose list an item is deleted.
item_count	Specifies the number of items to be deleted. This number must be nonnegative.
position	Specifies the position in the list of the first item to be deleted. A value of 1 indicates that the first deleted item is the first item in the list; a value of 2 indicates that it is the second item; and so on.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeletePos

Purpose

A List function that deletes an item from a list at a specified position

Synopsis

#include <Xm/List.h>

void XmListDeletePos(
  Widget *widget***,**
  int *position***);**

Description

XmListDeletePos deletes an item at a specified position. A warning message appears if the position does not exist.

widget	Specifies the ID of the List from which an item is to be deleted.
position	Specifies the position of the item to be deleted. A value of 1 indicates that the first item in the list is deleted; a value of 2 indicates that the second item is deleted; and so on. A value of 0 (zero) indicates that the last item in the list is deleted.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeletePositions

Purpose

A List function that deletes items from a list based on an array of positions

Synopsis

#include <Xm/List.h>

void XmListDeletePositions(
  Widget *widget***,**
  int ***position_list***,**
  int *position_count***);**

Description

XmListDeletePositions deletes noncontiguous items from a list. The function deletes all items whose corresponding positions appear in the *position_list* array. A warning message is displayed if a specified position is invalid; that is, the value is 0, a negative integer, or a number greater than the number of items in the list.

widget	Specifies the ID of the List widget
position_list	Specifies an array of the item positions to be deleted. The position of the first item in the list is 1; the position of the second item is 2; and so on.
position_count	Specifies the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeselectAllItems

Purpose

A List function that unhighlights and removes all items from the selected list

Synopsis

#include <Xm/List.h>

void XmListDeselectAllItems(
    Widget *widget***);**

Description

XmListDeselectAllItems unhighlights and removes all items from the selected list.

widget   Specifies the ID of the List widget from whose list all selected items are deselected

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeselectItem

Purpose

A List function that deselects the specified item from the selected list

Synopsis

#include <Xm/List.h>

void XmListDeselectItem(
    Widget *widget***,**
    XmString *item***);**

Description

XmListDeselectItem unhighlights and removes from the selected list the first item in the list that matches *item*.

widget	Specifies the ID of the List from whose list an item is deselected.
item	Specifies the item to be deselected from the list. If *item* appears more than once in the List, only the first occurrence is matched.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListDeselectPos

Purpose

A List function that deselects an item at a specified position in the list

Synopsis

#include <Xm/List.h>

void XmListDeselectPos(
    Widget *widget***,**
    int *position***);**

Description

XmListDeselectPos unhighlights the item at the specified position and deletes it from the list of selected items.

widget	Specifies the ID of the List widget
position	Specifies the position of the item to be deselected. A value of 1 indicates that the first item in the list is deselected; a value of 2 indicates that the second item is deselected; and so on. A value of 0 (zero) indicates that the last item in the list is deselected.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListGetKbdItemPos

Purpose

A List function that returns the position of the item at the location cursor

Synopsis

#include <Xm/List.h>

int XmListGetKbdItemPos(
    Widget *widget***);**

Description

XmListGetKbdItemPos returns the position of the list item at the location cursor.

widget	Specifies the ID of the List widget

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns the position of the current keyboard item. A value of 1 indicates that the location cursor is at the first item of the list; a value of 2 indicates that it is at the second item; and so on. A value of 0 (zero) indicates the List widget is empty.

Related Information

XmList(3).

XmListGetMatchPos

Purpose

A List function that returns all instances of an item in the list

Synopsis

#include <Xm/List.h>

Boolean XmListGetMatchPos(
    Widget *widget*,
    XmString *item*,
    **int *****position_list*,
    int ****position_count*);**

Description

XmListGetMatchPos is a Boolean function that returns an array of positions where a specified item is found in a List.

widget Specifies the ID of the List widget.

item Specifies the item to search for.

position_list Returns an array of positions at which the item occurs in the List. The position of the first item in the list is 1; the position of the second item is 2; and so on. When the return value is True, **XmListGetMatchPos** allocates memory for this array. The caller is responsible for freeing this memory. The caller can recover the allocated memory by calling **XtFree**.

position_count
Returns the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns True if the specified item is present in the list, and False if it is not.

Related Information

XmList(3).

XmListGetSelectedPos

Purpose

A List function that returns the position of every selected item in the list

Synopsis

#include <Xm/List.h>

Boolean XmListGetSelectedPos(
  Widget *widget***,**
  int ****position_list***,**
  int ***position_count***);**

Description

This routine is obsolete. It is replaced by calling **XtGetValues** for the List resources **XmNselectedPositions** and **XmNselectedPositionCount**. **XmListGetSelectedPos** is a Boolean function that returns an array of the positions of the selected items in a List.

widget Specifies the ID of the List widget.

position_list Returns an array of the positions of the selected items in the List. The position of the first item in the list is 1; the position of the second item is 2; and so on. When the return value is True, **XmListGetSelectedPos** allocates memory for this array. The caller is responsible for freeing this memory. The caller can recover the allocated memory by calling **XtFree**.

position_count
Returns the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns True if the list has any selected items, and False if it does not.

Related Information

XmList(3).

XmListItemExists

Purpose

A List function that checks if a specified item is in the list

Synopsis

```
#include <Xm/List.h>

Boolean XmListItemExists(
        Widget widget,
        XmString item);
```

Description

XmListItemExists is a Boolean function that checks if a specified item is present in the list.

widget	Specifies the ID of the List widget
item	Specifies the item whose presence is checked

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns True if the specified item is present in the list.

Related Information

XmList(3).

XmListItemPos

Purpose

A List function that returns the position of an item in the list

Synopsis

#include <Xm/List.h>

int XmListItemPos(
  Widget *widget***,**
  XmString *item***);**

Description

XmListItemPos returns the position of the first instance of the specified item in a list.

widget	Specifies the ID of the List widget
item	Specifies the item whose position is returned

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns the position in the list of the first instance of the specified item. The position of the first item in the list is 1; the position of the second item is 2; and so on. This function returns 0 (zero) if the item is not found.

Related Information

XmList(3).

XmListPosSelected

Purpose

A List function that determines if the list item at a specified position is selected

Synopsis

#include <Xm/List.h>

Boolean XmListPosSelected(
Widget *widget***,**
int *position***);**

Description

XmPosSelected determines if the list item at the specified position is selected or not.

widget	Specifies the ID of the List widget
position	Specifies the position of the list item. 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) specifies the last item in the list.

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns True if the list item is selected; otherwise, returns False if the item is not selected or the specified position is invalid.

Related Information

XmList(3).

XmListPosToBounds

Purpose

A List function that returns the bounding box of an item at a specified position in a list

Synopsis

#include <Xm/List.h>

Boolean XmListPosToBounds(
  Widget *widget***,**
  int *position***,**
  Position ****x*,**
  Position ****y*,**
  Dimension ****width*,**
  Dimension ****height*);**

Description

XmListPosToBounds returns the coordinates of an item within a list and the dimensions of its bounding box. The function returns 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, as well as the width and the height of the box. The caller can pass a NULL value for the *x*, *y*, *width*, or *height* parameters to indicate that the return value for that parameter is not requested.

widget	Specifies the ID of the List widget.
position	Specifies the position of the specified item. 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) specifies the last item in the list.
x	Specifies a pointer to the returned x-coordinate of the item.
y	Specifies the pointer to the returned y-coordinate of the item.
width	Specifies the pointer to the returned width of the item.
height	Specifies the pointer to the returned height of the item.

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

If the item at the specified position is not visible, returns False, and the returned values (if any) are undefined. Otherwise, this function returns True.

Related Information

XmList(3) and **XmListYToPos**(3).

XmListReplaceItems

Purpose

A List function that replaces the specified elements in the list

Synopsis

#include <Xm/List.h>

void XmListReplaceItems(
  Widget *widget***,**
  XmString ****old_items*,**
  int *item_count***,**
  XmString ****new_items*);**

Description

XmListReplaceItems replaces each specified item of the list with a corresponding new item. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget	Specifies the ID of the List widget.
old_items	Specifies the items to be replaced.
item_count	Specifies the number of items in *old_items* and *new_items*. This number must be nonnegative.
new_items	Specifies the replacement items.

Every occurrence of each element of *old_items* is replaced with the corresponding element from *new_items*. That is, the first element of *old_items* is replaced with the first element of *new_items*. The second element of *old_items* is replaced with the second element of *new_items*, and so on until *item_count* is reached.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListReplaceItemsPos

Purpose

A List function that replaces the specified elements in the list

Synopsis

#include <Xm/List.h>

void XmListReplaceItemsPos(
    Widget *widget***,**
    XmString ****new_items*,**
    int *item_count***,**
    int *position***);**

Description

XmListReplaceItemsPos replaces the specified number of items of the List with new items, starting at the specified position in the List. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget Specifies the ID of the List widget.

new_items Specifies the replacement items.

item_count Specifies the number of items in *new_items* and the number of items in the list to replace. This number must be nonnegative.

position Specifies the position of the first item in the list to be replaced. A value of 1 indicates that the first item replaced is the first item in the list; a value of 2 indicates that it is the second item; and so on.

Beginning with the item specified in *position*, *item_count* items in the list are replaced with the corresponding elements from *new_items*. That is, the item at *position* is replaced with the first element of *new_items*; the item after *position* is replaced with the second element of *new_items*; and so on, until *item_count* is reached.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListReplaceItemsPosUnselected

Purpose

A List function that replaces items in a list without selecting the replacement items

Synopsis

#include <Xm/List.h>

void XmListReplaceItemsPosUnselected(
Widget *widget***,**
XmString ****new_items*,**
int *item_count***,**
int *position***);**

Description

XmListReplaceItemsPosUnselected replaces the specified number of items in the list with new items, starting at the given position. The replacement items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget	Specifies the ID of the List widget to replace items in.
new_items	Specifies a pointer to the replacement items.
item_count	Specifies the number of elements in *new_items* and the number of items in the list to replace. This number must be nonnegative.
position	Specifies the position of the first item in the list to be replaced. A value of 1 indicates that the first item replaced is the first item in the list; a value of 2 indicates that it is the second item; and so on.
	Beginning with the item specified in *position*, *item_count* items in the list are replaced with the corresponding elements from *new_items*. That is, the item at *position* is replaced with the first element of *new_items*; the item after *position* is replaced with the second element of *new_items*; and so on, until *item_count* is reached.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListReplaceItemsUnselected

Purpose

A List function that replaces items in a list

Synopsis

#include <Xm/List.h>

void XmListReplaceItemsUnselected(
  Widget *widget***,**
  XmString ***old_items***,**
  int *item_count***,**
  XmString ***new_items***);**

Description

XmListReplaceItemsUnselected replaces each specified item in the list with a corresponding new item. The replacement items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget	Specifies the ID of the List widget to replace items in.
old_items	Specifies a pointer to the list items to be replaced.
item_count	Specifies the number of elements in *old_items* and *new_items*. This number must be nonnegative.
new_items	Specifies a pointer to the replacement items. Every occurrence of each element of *old_items* is replaced with the corresponding element from *new_items*. That is, the first element of *old_items* is replaced with the first element of *new_items*. The second element of *old_items* is replaced with the second element of *new_items*, and so on until *item_count* is reached. If an element in *old_items* does not exist in the list, the corresponding entry in *new_items* is skipped.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListReplacePositions

Purpose

A List function that replaces items in a list based on position

Synopsis

#include <Xm/List.h>

void XmListReplacePositions(
    Widget *widget***,**
    int ***position_list*,**
    XmString ***item_list*,**
    int *item_count;***);**

Description

XmListReplacePositions replaces noncontiguous items in a list. The item at each position specified in *position_list* is replaced with the corresponding entry in *item_list*. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. Any of the new items that match items on the selected list appear selected. A warning message is displayed if a specified position is invalid; that is, the value is 0 (zero), a negative integer, or a number greater than the number of items in the list.

widget	Specifies the ID of the List widget.
position_list	Specifies an array of the positions of items to be replaced. The position of the first item in the list is 1; the position of the second item is 2; and so on.
item_list	Specifies an array of the replacement items.
item_count	Specifies the number of elements in *position_list* and *item_list*. This number must be nonnegative.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListSelectItem

Purpose

A List function that selects an item in the list

Synopsis

#include <Xm/List.h>

void XmListSelectItem(
Widget *widget***,**
XmString *item***,**
Boolean *notify***);**

Description

XmListSelectItem highlights and adds to the selected list the first item in the list that matches *item*.

widget	Specifies the ID of the List widget from whose list an item is selected.
item	Specifies the item to be selected in the List widget. If *item* appears more than once in the List, only the first occurrence is matched.
notify	Specifies a Boolean value that when TRUE invokes the selection callback for the current mode. From an application interface view, calling this function with *notify* True is indistinguishable from a user-initiated selection action. When *notify* is FALSE, no callbacks are called.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3) and **XmListSelectPos**(3).

XmListSelectPos

Purpose

A List function that selects an item at a specified position in the list

Synopsis

#include <Xm/List.h>

void XmListSelectPos(
    Widget *widget***,**
    int *position***,**
    Boolean *notify***);**

Description

XmListSelectPos highlights a List item at the specified position and adds it to the list of selected items.

widget	Specifies the ID of the List widget.
position	Specifies the position of the item to be selected. A value of 1 indicates that the first item in the list is selected; a value of 2 indicates that the second item is selected; and so on. A value of 0 (zero) indicates that the last item in the list is selected.
notify	Specifies a Boolean value that when TRUE invokes the selection callback for the current mode. From an application interface view, calling this function with *notify* True is indistinguishable from a user-initiated selection action. When *notify* is FALSE, no callbacks are called.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3) and **XmListSelectItem**(3).

XmListSetAddMode

Purpose

A List function that sets add mode in the list

Synopsis

#include <Xm/List.h>

void XmListSetAddMode(
 Widget *widget***,**
 Boolean *state***);**

Description

XmListSetAddMode allows applications control over Add Mode in the extended selection model. This function ensures that the mode it sets is compatible with the selection policy (**XmNselectionPolicy**) of the widget. For example, it cannot put the widget into add mode when the value of **XmNselectionPolicy** is **XmBROWSE_SELECT**.

widget	Specifies the ID of the List widget
state	Specifies whether to activate or deactivate Add Mode. If *state* is True, Add Mode is activated. If *state* is False, Add Mode is deactivated.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListSetBottomItem

Purpose

A List function that makes an existing item the last visible item in the list

Synopsis

#include <Xm/List.h>

void XmListSetBottomItem(
    Widget *widget***,**
    XmString *item***);**

Description

XmListSetBottomItem makes the first item in the list that matches *item* the last visible item in the list.

widget	Specifies the ID of the List widget from whose list an item is made the last visible
item	Specifies the item

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListSetBottomPos

Purpose

A List function that makes a specified item the last visible item in the list

Synopsis

#include <Xm/List.h>

void XmListSetBottomPos(
    Widget *widget***,**
    int *position***);**

Description

XmListSetBottomPos makes the item at the specified position the last visible item in the List.

widget	Specifies the ID of the List widget.
position	Specifies the position of the item to be made the last visible item in the list. A value of 1 indicates that the first item in the list is the last visible item; a value of 2 indicates that the second item is the last visible item; and so on. A value of 0 (zero) indicates that the last item in the list is the last visible item.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListSetHorizPos

Purpose

A List function that scrolls to the specified position in the list

Synopsis

#include <Xm/List.h>

void XmListSetHorizPos(
Widget *widget***,**
int *position***);**

Description

XmListSetHorizPos sets the **XmNvalue** resource of the horizontal ScrollBar to the specified position and updates the visible portion of the list with the new value if the List widget's **XmNlistSizePolicy** is set to **XmCONSTANT** or **XmRESIZE_IF_POSSIBLE** and the horizontal ScrollBar is currently visible. This is equivalent to moving the horizontal ScrollBar to the specified position.

widget	Specifies the ID of the List widget
position	Specifies the horizontal position

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListSetItem

Purpose

A List function that makes an existing item the first visible item in the list

Synopsis

#include <Xm/List.h>

void XmListSetItem(
    Widget *widget***,**
    XmString *item***);**

Description

XmListSetItem makes the first item in the list that matches *item* the first visible item in the list.

widget	Specifies the ID of the List widget from whose list an item is made the first visible
item	Specifies the item

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListSetKbdItemPos

Purpose

A List function that sets the location cursor at a specified position

Synopsis

#include <Xm/List.h>

Boolean XmListSetKbdItemPos(
    Widget *widget***,**
    int *position***);**

Description

XmListSetKbdItemPos sets the location cursor at the item specified by *position*. This function does not determine if the item at the specified position is selected or not.

widget Specifies the ID of the List widget.

position Specifies the position of the item at which the location cursor is set. 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) sets the location cursor at the last item in the list.

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns False if no item exists at the specified position or if the list is empty; otherwise, returns True.

Related Information

XmList(3).

XmListSetPos

Purpose

A List function that makes the item at the given position the first visible position in the list

Synopsis

#include <Xm/List.h>

void XmListSetPos(
  Widget *widget***,**
  int *position***);**

Description

XmListSetPos makes the item at the given position the first visible position in the list.

widget	Specifies the ID of the List widget.
position	Specifies the position of the item to be made the first visible item in the list. A value of 1 indicates that the first item in the list is the first visible item; a value of 2 indicates that the second item is the first visible item; and so on. A value of 0 (zero) indicates that the last item in the list is the first visible item.

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListUpdateSelectedList

Purpose

A List function that updates the XmNselectedItems resource

Synopsis

```
#include <Xm/List.h>

void XmListUpdateSelectedList(
        Widget widget);
```

Description

XmListUpdateSelectedList frees the contents of the current **XmNselectedItems** list. The routine traverses the **XmNitems** list and adds each currently selected item to the **XmNselectedItems** list. For each selected item, there is a corresponding entry in the updated **XmNselectedItems** list.

widget	Specifies the ID of the List widget to update

For a complete definition of List and its associated resources, see **XmList**(3).

Related Information

XmList(3).

XmListYToPos

Purpose

A List function that returns the position of the item at a specified y-coordinate

Synopsis

```
#include <Xm/List.h>

int XmListYToPos(
        Widget widget,
        Position y);
```

Description

XmListYToPos returns the position of the item at the given y-coordinate within the list.

widget	Specifies the ID of the List widget
y	Specifies the y-coordinate in the list's coordinate system

For a complete definition of List and its associated resources, see **XmList**(3).

Return Values

Returns the position of the item at the specified y coordinate. 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.

Related Information

XmList(3) and **XmListPosToBounds**(3).

XmMainWindowSep1

Purpose

A MainWindow function that returns the widget ID of the first Separator

Synopsis

#include <Xm/MainW.h>

Widget XmMainWindowSep1(
  Widget *widget***);**

Description

XmMainWindowSep1 returns the widget ID of the first Separator in the MainWindow. The first Separator is located between the MenuBar and the Command widget. This Separator is visible only when **XmNshowSeparator** is True.

NOTE: XmMainWindowSep1 is obsolete and exists for compatibility with previous releases. Use **XtNameToWidget** instead. Pass a MainWindow variable as the first argument to **XtNameToWidget** and pass **Separator1** as the second argument.

widget	Specifies the MainWindow widget ID

For a complete definition of MainWindow and its associated resources, see **XmMainWindow**(3).

Return Values

Returns the widget ID of the first Separator.

Related Information

XmMainWindow(3).

XmMainWindowSep2

Purpose

A MainWindow function that returns the widget ID of the second Separator widget

Synopsis

#include <Xm/MainW.h>

Widget XmMainWindowSep2(
    Widget *widget***);**

Description

XmMainWindowSep2 returns the widget ID of the second Separator in the MainWindow. The second Separator is located between the Command widget and the ScrolledWindow. This Separator is visible only when **XmNshowSeparator** is True.

NOTE: XmMainWindowSep2 is obsolete and exists for compatibility with previous releases. Use **XtNameToWidget** instead. Pass a MainWindow variable as the first argument to **XtNameToWidget** and pass **Separator2** as the second argument.

widget	Specifies the MainWindow widget ID

For a complete definition of MainWindow and its associated resources, see **XmMainWindow**(3).

Return Values

Returns the widget ID of the second Separator.

Related Information

XmMainWindow(3).

XmMainWindowSep3

Purpose

A MainWindow function that returns the widget ID of the third Separator widget

Synopsis

#include <Xm/MainW.h>

Widget XmMainWindowSep3(
    Widget *widget***);**

Description

XmMainWindowSep3 returns the widget ID of the third Separator in the MainWindow. The third Separator is located between the message window and the widget above it. This Separator is visible only when **XmNshowSeparator** is True.

NOTE: XmMainWindowSep3 is obsolete and exists for compatibility with previous releases. Use **XtNameToWidget** instead. Pass a MainWindow variable as the first argument to **XtNameToWidget** and pass **Separator3** as the second argument.

widget	Specifies the MainWindow widget ID

For a complete definition of MainWindow and its associated resources, see **XmMainWindow**(3).

Return Values

Returns the widget ID of the third Separator.

Related Information

XmMainWindow(3).

XmMainWindowSetAreas

Purpose

A MainWindow function that identifies manageable children for each area

Synopsis

#include <Xm/MainW.h>

void XmMainWindowSetAreas(
  Widget *widget***,**
  Widget *menu_bar***,**
  Widget *command_window***,**
  Widget *horizontal_scrollbar***,**
  Widget *vertical_scrollbar***,**
  Widget *work_region***);**

Description

XmMainWindowSetAreas identifies which of the valid children for each area (such as the MenuBar and work region) are to be actively managed by MainWindow. This function also sets up or adds the MenuBar, work window, command window, and ScrollBar widgets to the application's main window widget.

Each area is optional; therefore, the user can pass NULL to one or more of the following arguments. The window manager provides the title bar.

NOTE: XmMainWindowSetAreas is obsolete and exists for compatibility with previous releases. The information previously returned by this function can now be obtained through a call to **XtGetValues** on the **XmNscrolledWindowChildType** resource.

widget	Specifies the MainWindow widget ID.
menu_bar	Specifies the widget ID for the MenuBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNmenuBar**.

command_window
Specifies the widget ID for the command window to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNcommandWindow**.

horizontal_scrollbar
Specifies the ScrollBar widget ID for the horizontal ScrollBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNhorizontalScrollBar**.

vertical_scrollbar
Specifies the ScrollBar widget ID for the vertical ScrollBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNverticalScrollBar**.

work_region
Specifies the widget ID for the work window to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNworkWindow**.

For a complete definition of MainWindow and its associated resources, see **XmMainWindow**(3).

Related Information

XmMainWindow(3).

XmMapSegmentEncoding

Purpose

A compound string function that returns the compound text encoding format associated with the specified font list tag

Synopsis

#include <Xm/Xm.h>

char * XmMapSegmentEncoding(
char ****fontlist_tag*);**

Description

XmMapSegmentEncoding searches the segment encoding registry for an entry that matches the specified font list tag and returns a copy of the associated compound text encoding format. The application is responsible for freeing the storage associated with the returned data by calling **XtFree**.

fontlist_tag Specifies the compound string font list tag

Return Values

Returns a copy of the associated compound text encoding format if the font list tag is found in the registry; otherwise, returns NULL.

Related Information

XmCvtXmStringToCT(3), **XmFontList**(3), **XmRegisterSegmentEncoding**(3), and **XmString**(3).

XmMenuPosition

Purpose

A RowColumn function that positions a Popup menu pane

Synopsis

#include <Xm/RowColumn.h>

void XmMenuPosition(
        Widget *menu***,**
        XButtonPressedEvent * *event***);**

Description

XmMenuPosition positions a Popup menu pane using the information in the specified event. Unless an application is positioning the menu pane itself, it must first invoke this function before managing the PopupMenu. The *x_root* and *y_root* fields in the specified X event are used to determine the menu position.

menu	Specifies the PopupMenu to be positioned
event	Specifies the event passed to the action procedure which manages the PopupMenu

Which corner of the PopupMenu is positioned at the *x_root* and *y_root* depends on the **XmNlayoutDirection** resource of the widget from which popup occurs.

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Related Information

XmRowColumn(3).

XmMessageBoxGetChild

Purpose

A MessageBox function that is used to access a component

Synopsis

#include <Xm/MessageB.h>

Widget XmMessageBoxGetChild(
    Widget *widget***,**
    unsigned char *child***);**

Description

XmMessageBoxGetChild is used to access a component within a MessageBox. The parameters given to the function are the MessageBox widget and a value indicating which component to access.

NOTE: This routine is obsolete and exists for compatibility with previous releases. Instead of calling **XmMessageBoxGetChild**, you should call **XtNameToWidget** as described in the **XmMessageBox**(3) reference page.

widget Specifies the MessageBox widget ID.

child Specifies a component within the MessageBox. The following are legal values for this parameter:

- **XmDIALOG_CANCEL_BUTTON**
- **XmDIALOG_DEFAULT_BUTTON**
- **XmDIALOG_HELP_BUTTON**
- **XmDIALOG_MESSAGE_LABEL**
- **XmDIALOG_OK_BUTTON**
- **XmDIALOG_SEPARATOR**
- **XmDIALOG_SYMBOL_LABEL**

For a complete definition of MessageBox and its associated resources, see **XmMessageBox**(3).

Return Values

Returns the widget ID of the specified MessageBox component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmMessageBox(3).

XmNotebookGetPageInfo

Purpose

A Notebook function that returns page information

Synopsis

#include <Xm/Notebook.h>

XmNotebookPageStatus XmNotebookGetPageInfo(
  Widget *notebook***,**
  int *page_number***,**
  XmNotebookPageInfo **page_info***);**

Description

XmNotebookGetPageInfo returns status information for the specified Notebook page.

notebook Specifies the Notebook widget.

page_number
Specifies the page number to be queried.

page_info Points to the structure containing the page information. The structure has the following form:

```
typedef struct
{
        int      page_number;
        Widget   page_widget;
        Widget   status_area_widget;
        Widget   major_tab_widget;
        Widget   minor_tab_widget;
} XmNotebookPageInfo;
```

page_number
Specifies the *page_number* passed to the function.

page_widget
Specifies a child widget of the Notebook with a **XmNchildType** of **XmPAGE** and a **XmNpageNumber**

equal to *page_number* if one exists; otherwise set to NULL.

status_area_widget
: Specifies a child widget of the Notebook with a **XmNchildType** of **XmSTATUS_AREA** and a **XmNpageNumber** equal to *page_number* if one exists; otherwise set to NULL.

major_tab_widget
: Specifies a child widget of the Notebook with a **XmNchildType** of **XmMAJOR_TAB** and the nearest **XmNpageNumber** equal to or less than *page_number* if one exists; otherwise set to NULL.

minor_tab_widget
: Specifies a child widget of the Notebook with a **XmNchildType** of **XmMINOR_TAB** and the nearest **XmNpageNumber** equal to or less than *page_number* if one exists; otherwise set to NULL.

For a complete definition of Notebook and its associated resources, see **XmNotebook**(3).

Return Values

Returns one of the following page status values:

XmPAGE_FOUND
: The specified page was found.

XmPAGE_INVALID
: The specified page number is out of the page number range.

XmPAGE_EMPTY
: The specified page does not have a page widget.

XmPAGE_DUPLICATED
: There is more than one page widget with the specified page number. The more recently managed page widget is used for the page information structure.

Related Information

XmNotebook(3).

XmObjectAtPoint

Purpose

A toolkit function that determines which child intersects or comes closest to a specified point

Synopsis

```
#include <Xm/Xm.h>

Widget XmObjectAtPoint(
        Widget widget,
        Position x,
        Position y);
```

Description

XmObjectAtPoint searches the child list of the specified manager *widget* and returns the child most closely associated with the specified *x*,*y* coordinate pair.

For the typical Motif manager *widget*, **XmObjectAtPoint** uses the following rules to determine the returned object:

- If one child intersects *x*,*y*, **XmObjectAtPoint** returns the widget ID of that child.
- If more than one child intersects *x*,*y*, **XmObjectAtPoint** returns the widget ID of the visible child.
- If no child intersects *x*,*y*, **XmObjectAtPoint** returns NULL.

The preceding rules are only general. In fact, each manager *widget* is free to define "most closely associated" as it desires. For example, if no child intersects *x*,*y*, a manager might return the child closest to *x*,*y*.

widget	Specifies a manager widget.
x	Specifies the x-coordinate about which you are seeking child information. The x-coordinate must be specified in pixels, relative to the left side of *manager*.

y Specifies the y-coordinate about which you are seeking child information. The y-coordinate must be specified in pixels, relative to the top side of *manager*.

Return Values

Returns the child of *manager* most closely associated with *x*,*y*. If none of its children are sufficiently associated with *x*,*y*, returns NULL.

Related Information

XmManager(3).

XmOptionButtonGadget

Purpose

A RowColumn function that obtains the widget ID for the CascadeButtonGadget in an OptionMenu

Synopsis

```
#include <Xm/RowColumn.h>

Widget XmOptionButtonGadget(
        Widget option_menu);
```

Description

XmOptionButtonGadget provides the application with the means for obtaining the widget ID for the internally created CascadeButtonGadget. Once the application has obtained the widget ID, it can adjust the visuals for the CascadeButtonGadget, if desired.

When an application creates an instance of the OptionMenu widget, the widget creates two internal gadgets. One is a LabelGadget that is used to display RowColumn's **XmNlabelString** resource. The other is a CascadeButtonGadget that displays the current selection and provides the means for posting the OptionMenu's submenu.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas.

Option Menu Label Gadget
: **OptionLabel**

Option Menu Cascade Button
: **OptionButton**

option_menu Specifies the OptionMenu widget ID

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the widget ID for the internal button.

Related Information

XmCreateOptionMenu(3), **XmCascadeButtonGadget**(3), **XmOptionLabelGadget**(3), and **XmRowColumn**(3).

XmOptionLabelGadget

Purpose

A RowColumn function that obtains the widget ID for the LabelGadget in an OptionMenu

Synopsis

#include <Xm/RowColumn.h>

Widget XmOptionLabelGadget(
          Widget *option_menu***);**

Description

XmOptionLabelGadget provides the application with the means for obtaining the widget ID for the internally created LabelGadget. Once the application has obtained the widget ID, it can adjust the visuals for the LabelGadget, if desired.

option_menu Specifies the OptionMenu widget ID

When an application creates an instance of the OptionMenu widget, the widget creates two internal gadgets. One is a LabelGadget that is used to display RowColumn's **XmNlabelString** resource. The other is a CascadeButtonGadget that displays the current selection and provides the means for posting the OptionMenu's submenu.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas.

Option Menu Label Gadget
          OptionLabel

Option Menu Cascade Button
          OptionButton

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the widget ID for the internal label.

Related Information

XmCreateOptionMenu(3), **XmLabelGadget**(3), **XmOptionButtonGadget**(3), and **XmRowColumn**(3).

XmParseMappingCreate

Purpose

A compound string function to create a parse mapping

Synopsis

#include <Xm/Xm.h>

XmParseMapping XmParseMappingCreate(
        ArgList *arglist***,**
        Cardinal *argcount***);**

Description

XmParseMappingCreate creates a parse mapping for use in a parse table. This function allows the application to specify values for components of the parse mapping using a resource-style argument list.

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of **XmParseMapping** and its associated resources, see **XmParseMapping**(3).

Return Values

Returns the **XmParseMapping** object. The function allocates space to hold the returned **XmParseMapping** object. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmParseMappingFree**.

Related Information

XmParseMapping(3), **XmParseMappingFree**(3), **XmParseMappingGetValues**(3), **XmParseMappingSetValues**(3), **XmParseTable**(3), and **XmString**(3).

XmParseMappingFree

Purpose

A compound string function to free a parse mapping

Synopsis

#include <Xm/Xm.h>

void XmParseMappingFree(
    XmParseMapping *parse_mapping***);**

Description

XmParseMappingFree recovers memory used by an **XmParseMapping**.

parse_mapping
Specifies the parse mapping to be freed

Related Information

XmParseMapping(3), **XmParseMappingCreate**(3), **XmParseMappingGetValues**(3), **XmParseMappingSetValues**(3), **XmParseTable**(3), and **XmString**(3).

XmParseMappingGetValues

Purpose

A compound string function to retrieve attributes of a parse mapping

Synopsis

#include <Xm/Xm.h>

void XmParseMappingGetValues(
  XmParseMapping *parse_mapping***,**
  ArgList *arglist***,**
  Cardinal *argcount***);**

Description

XmParseMappingGetValues retrieves attributes of an **XmParseMapping** object, using a resource-style argument list. If the **XmNsubstitute** resource is in the *arglist*, the function will allocate space to hold the returned **XmString** value. The application is responsible for managing this allocated space. The application can recover the allocated space by calling **XmStringFree**.

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of **XmParseMapping** and its associated resources, see **XmParseMapping**(3).

Related Information

XmParseMapping(3), **XmParseMappingCreate**(3), **XmParseMappingFree**(3), **XmParseMappingSetValues**(3), **XmParseTable**(3), and **XmString**(3).

XmParseMappingSetValues

Purpose

A compound string function to set attributes of a parse mapping

Synopsis

```
#include <Xm/Xm.h>

void XmParseMappingSetValues(
        XmParseMapping parse_mapping,
        ArgList arglist,
        Cardinal argcount);
```

Description

XmParseMappingSetValues specifies attributes of an **XmParseMapping** object, using a resource-style argument list.

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of **XmParseMapping** and its associated resources, see **XmParseMapping**(3).

Related Information

XmParseMapping(3), **XmParseMappingCreate**(3), **XmParseMappingFree**(3), **XmParseMappingGetValues**(3), **XmParseTable**(3), and **XmString**(3).

XmParseTableFree

Purpose

A compound string function that recovers memory

Synopsis

```
#include <Xm/Xm.h>

void XmParseTableFree(
        XmParseTable parse_table,
        Cardinal count);
```

Description

XmParseTableFree recovers memory used by an **XmParseTable** and its constituent **XmParseMapping**s.

parse_table	Specifies the parse table to be freed
count	Specifies the number of parse mappings in the parse table

Related Information

XmParseTable(3) and **XmString**(3).

XmGetScaledPixmap

Purpose

read a pixmap file and scale it according to pixmap and print resolution

Synopsis

#include <Xm/Xm.h>

XtEnum XmGetScaledPixmap(
  Widget *widget***,**
  String *image_name***,**
  Pixel *foreground***,**
  Pixel *background***,**
  int *depth***,**
  Double *scaling_ratio***);**

Description

XmGetScaledPixmap uses its *Widget* argument to look up for a Print Shell ancestor to get the pixmap resolution and the default printer resolution information to be used if *scaling_ratio* **==0**.

If scaling is 0, and a valid PrintShell is present **XmGetScaledPixmap** applies a ratio equals to (printer resolution / default pixmap resolution) before creating the Pixmap on the widget’s Screen. Otherwise, the *scaling_ratio* is used in scaling both dimensions of the image being converted as a Pixmap.

XmGetScaledPixmap completes the **XmGetPixmapByDepth** existing API by making use of the *XmNdefaultPixmapResolution* of the rooting **XmPrintShell**. Refer to the **XmGetPixmapByDepth** documentation for details.

widget	Widget used to determine the default pixmap resolution (of the print shell ancestor).
image_name	See XmGetPixmapByDepth for description.
foreground	See XmGetPixmapByDepth for description.
background	See XmGetPixmapByDepth for description.

depth See XmGetPixmapByDepth for description.

scaling_ratio
Indicate the scaling ratio to be applied, or 0.

Return Values

Returns Pixmap or NULL if failed.

Errors/Warnings

Same as for **XmGetPixmapByDepth**.

Related Information

XmPrintSetup(3), **XmPrintShell**(3), **XmRedisplayWidget**(3)

XmPrintPopupPDM

Purpose

Send a notification for the PDM to be popped up

Synopsis

#include <Xm/Print.h>

XtEnum XmPrintPopupPDM(
    Widget*print_shell***,**
    Widget*video_transient_for***);**

Description

A convenience function that sends a notification to start a Print Dialog Manager on behalf of the application, **XmPrintPopupPDM** hides the details of the X selection mechanism used to notify the PDM that a new dialog must be popped up for this application.

XmPrintPopupPDM sends a selection request to either the print display of the print shell, or the video display of the transient_for video widget (depending on the environment variable *XPDMDISPLAY*, which can only takes the value "print" or "video"), asking for the PDM windows to be popped up on behalf of the app.

Return right away with status of *XmPDM_NOTIFY_FAIL* (e.g. if the function couldn't malloc memory for the selection value, or if *XPDMDISPLAY* is not "print" or "video") or with *XmPDM_NOTIFY_SUCCESS* , which only means a "message" was sent out to the PDM specified by *XPDMSELECTION* , not that it's already up on the screen yet.

In order to know if the PDM is up, or not running, the application must register a **XmNpdmNotificationCallback** with the Print Shell.

XmPrintPopupPDM puts up an **InputOnly** window on top of the dialog, so that the end user doesn't use the print setup dialog while the PDM is trying to come up. This window is automatically removed when the shell is about to call the callback for the first time.

print_shell   The Print Shell used for this print job and context.

video_transient_for
The video widget dealing with application print setup.

Return Values

Returns *XmPDM_NOTIFY_SUCCESS* if the function was able to send the notification out to the PDM process, *XmPDM_NOTIFY_FAIL* otherwise.

Errors/Warnings

Not applicable.

Examples

Example of callback from a Print set up dialog box "Setup..." button:

```
PrintSetupCallback(print_dialog...)
/*-------------*/
{
    if (XmPrintPopupPDM (pshell, XtParent(print_dialog)) !=
                                    XmPDM_NOTIFY_SUCCESS) {
        /* some error dialog */
    }
}
```

Example of **XmNpdmNotificationCallback** from a Print Shell:

```
pdmNotifyCB(print_shell...)
{
    XmPrintShellCallBackStruct * pr_cb = ...

    switch (pr_cb->reason) {
       case XmCR_PDM_NONE:
           /* no PDM available */
           PostErrorDialog(...);
           break;
       case XmCR_PDM_VXAUTH:
```

```
                  /* PDM is not authorized ... */
                  PostErrorDialog(...);
                  break;
              case XmCR_PDM_UP: the PDM is up and running
                  /* everything is fine */
                  break;
                      default: /* other cases */
          }
       }
```

Related Information

XmPrintSetup(3), **XmPrintShell**(3), **XmRedisplayWidget**(3), **XmPrintToFile**(3)

XmPrintSetup

Purpose

setup and create a Print Shell widget

Synopsis

#include <Xm/Print.h>

Widget XmPrintSetup(
  Widget *video_widget***,**
  Screen **print_screen***,**
  String *print_shell_name***,**
  ArgList *args***,**
  Cardinal *num_args***);**

Description

A function that does the appropriate setting and creates a realized *XmPrintShell* that it returns to the caller. This function hides the details of the **Xt** to set up a valid print shell heirarchy for the application. It is also meant to encourage consistency in the way applications root their print widget hierarchy.

print_screen must belong to a Display connection that has already been initialized with **Xt**.

The *video_widget* is used to get at the application context, application name and class, and **argc/argv** stored on the **applicationShell** that roots this widget. If no **applicationShell** is found, **NULL argv/argc** are used.

XmPrintSetup then creates an unrealized **ApplicationShell** with the same name and class as the one given by the video display, on the print display and on the print screen specified.

An *XmPrintShell* is then created as a child of this toplevel shell, using **XtCreatePopupShell**, with the name *print_shell_name*, and using the *args* provided. It then realizes and maps the print shell, using *XtPopup* with *XtGrabNone*.

This way, application resource files and users can specify print specific attributes using the following syntax (if **print_shell_name** is "Print"):

```
Dtpad.Print*textFontList: somefont
*Print*background:white
*Print*highlightThickness:0
```

video_widget
A video widget to fetch app video data from.

print_screen A print screen on the print display - specifies the screen onto which the new shell is created.

print_shell_name
Specifies the name of the XmPrintShell created on the X Print server.

args Specifies the argument list from which to get the resources for the XmPrintShell.

num_args Specifies the number of arguments in the argument list.

Return Values

The id the *XmPrintShell* widget created on the X Print Server connection, or NULL if an error has occured.

Errors/Warnings

None.

Examples

From the **OK** callback and the **SetUp** callback of the primary print dialog widget:

```
static void
printOKCB(Widget, XtPointer call_data, XtPointer client_data)
{
  AppPrint *p = (AppPrint *) client_data;
  DtPrintSetupCallbackStruct *pbs =
                         (XmPrintCallbackStruct *) call_data;

  /* connect if not already done.
```

```
     the print dialog callback always provides valid
             printer name, print display and screen
             already initialized: XpInitContext called */
 */
  p->print_shell = XmPrintSetup (widget, pbs->print_screen,
                                         "Print", NULL, 0);

  ...
}
```

Related Information

XmPrintShell(3), **XmRedisplayWidget**(3), **XmPrintToFile**(3), **XmPrintPopupPDM**(3)

XmPrintShell

Purpose

a shell widget class used for printing in Motif

Synopsis

#include <Xm/Print.h>

```
Boolean XmIsPrintShell(
          Widget);
```

Description

The **XmPrintShell** provides the Motif application programmer with an Xt widget oriented API to some of the X Print resources and a callback to drive the pagination.

The **XmPrintShell** provides a simple callback to handle the pagination logic, and a set of resources to get and set common printer attributes.

If not created on an **XPrint** connection, **XmPrintShell** behaves as a regular applicationShell.

The **XmPrintShell** also initializes the **Xp** extension event handling mechanism, by registering an extension selector that calls **XpSelectInput** and event dispatcher for print and attributes **Xp** events, so applications can use **XtInsertEventTypeHandler** to register their own handler with the **Xp** events.

Arguments

No **XmCreate** function is provided, since this is a toplevel shell, most likely created thru some **Xt** shell creation routine or **XmPrintSetup**.

Classes

XmPrintShell is a subclass of **ApplicationShell**; it inherits behavior, resources and traits from all its superclasses. The class pointer is *XmPrintShellWidgetClass*.

New Resources

XmPrintShell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNstartJobCallback	*XmCCallback*	*XtCallbackList*	*NULL*	**CSG**
XmNendJobCallback	*XmCCallback*	*XtCallbackList*	*NULL*	**CSG**
XmNpageSetupCallback	*XmCCallback*	*XtCallbackList*	*NULL*	**CSG**
XmNminX	*XmCMinX*	**Dimension**	**dynamic**	**G**
XmNminY	*XmCMinY*	**Dimension**	**dynamic**	**G**
XmNmaxX	*XmCMaxX*	**Dimension**	**dynamic**	**G**
XmNmaxY	*XmCMaxY*	**Dimension**	**dynamic**	**G**
XmNdefaultPixmap-Resolution	*XmCDefaultPixmap-Resolution*	**unsigned short**	**100**	**CSG**
XmNpdmNotification-Callback	*XmCCallback*	*XtCallbackList*	*NULL*	**CSG**

XmNstartJobCallback
Specifies the callback driving the beginning of rendering. It is safe for an application to start rendering after this callback has been activated. **XpStartJob** must be called to trigger this callback.

XmNendJobCallback
Specifies the callback driving the end of rendering. Notify the client that all rendering has been processed (whether on print-to-file or regular spool). **XpEndJob** is called by the print shell to trigger this callback.

XmNpageSetupCallback
Specifies the callback driving the page layout. It is safe for an app to start rendering from this callback even if the **XmNstartJobCallback** is not used.

XmNminX, XmNminY, XmNmaxX, XmNmaxY
Specify the imageable area of the page in the current print context. **XmPrintShell** also maintains a proper size at all times by updating its own widget dimension whenever an attribute, such as resolution or orientation, changes. It is sized in its **Initialize** routine so that the application can rely on a proper size before the first **StartPage** call is issued.

XmNdefaultPixmapResolution

Indicates the resolution in dpi (dot per inch) of the image files read and converted by Motif for the widget descendants of this shell. It is used to determine a scaling ratio to be applied to pixmap created thru regular pixmap/icon conversion of the following Widget resources:

- *XmLabel*.label*Pixmap, *XmIconG*.*IconPixmap *XmToggleB*.selectPixmap, *XmPushBG*.armPixmap, *XmIconG*.*IconMask, *XmMessageBox*.symbolPixmap, *XmContainer*.*StatePixmap, ...
- Leaving out the pixmap resources being used for tiling (XmNhighlightPixmap, XmNtopShadowPixmap, XmNbottomShadowPixmap, XmNbackgroundPixmap, ...)

XmNpdmNotificationCallback

A callback notifying the application about the status of the PDM (see XmPrintPopupPDM). A XmPrintShellCallbackStruct is used, with reason:

- *XmCR_PDM_NONE*: no PDM available on this display for the named selection (provided in detail)
- *XmCR_PDM_START_VXAUTH* : the PDM is not authorized to connect to the video display.
- *XmCR_PDM_START_PXAUTH* : the PDM is not authorized to connect to the print display.
- *XmCR_PDM_UP* : the PDM is up and running
- *XmCR_PDM_OK* : the PDM has exited with OK status
- *XmCR_PDM_CANCEL* : the PDM has exited with CANCEL
- *XmCR_PDM_START_ERROR* : the PDM cannot start due to some error (usually logged)
- *XmCR_PDM_EXIT_ERROR* : the PDM has exited with an error

Callback Information

The **XmNstartJobCallback**, **XmNendJobCallback, XmNpageSetupCallback** and **XmNpdmNotificationCallback** operate on a *XmPrintShellCallbackStruct*, which is defined as follow:

```
typedef struct
{
    int     reason;  /* XmCR_START_JOB, XmCR_END_JOB,
                        XmCR_PAGE_SETUP, XmCR_PDM_* */
    XEvent  *event;
    XPContext print_context;
    Boolean last_page; /* in_out */
    XtPointer detail;
} XmPrintShellCallbackStruct;
```

Additional Behavior

The *last_page* field is only meaningful when the reason is *XmCR_PAGE_SETUP*.

The page setup callback is called with *last_page* **False** to notify the application that it has to get its internal layout state ready for the next page. Typically, a widget based application will change the content of a **Label** showing the page number, or scroll the content of the **Text** widget.

When the application has processed its last page, it should set the *last_page* field in the callback struct to **True**. The callback will be called a last time after that with *last_page* **False** to notify the application that it can safely clean-up its internal state (e.g., destroy widgets).

No drawing should occur from within the callback function in the application, this is an Exposure event-driven programming model where widgets render themselves from their expose methods.

The print shell calls **XpStartPage** after the **pageSetupCallback** returns, and **XpEndPage** upon reception of **StartPageNotify**.

Errors/Warnings

XmPrintShell can generate the following warnings:

- **Not connected to a valid X Print Server: behavior undefined.**
- **Attempt to set an invalid resolution on a printer: %s**
- **Attempt to set an invalid orientation on a printer: %s**

Return Values

Not applicable

Examples

```
PrintOnePageCB(Widget pshell, XtPointer npages,
/*----------*/ XmPrintSetPageCBStruct psp)
{
    static int cur_page = 0;
    cur_page++;

    if (! psp->last_page
        && curPage > 1) /* no need to scroll for the first page */
    {

        XmTextScroll(ptext, prows);  /* get ready for next page */

    } else {    /**** I'm done */

       XtDestroyWidget(pshell);
       XtCloseDisplay(XtDisplay(pshell));
    }

    if (cur_page == (int) n_pages) psp->last_page = True;
}

PrintOKCallback(...)
/*-------------*/
{
    pshell = XmPrintSetup (widget, pbs->print_screen,
                                   "Print", NULL, 0);

    XpStartJob(XtDisplay(pshell), XPSpool);

    /**** here I get the size of the shell, create my widget
          hierarchy: a bulleting board, and then a text widget,
          that I stuff with the video text widget buffer */
    /* get the total number of pages to print */
    /* same code as previous example to get n_pages */

    /****  set up my print callback */
    XtAddCallback(pshell,  XmNpageSetUpCallback,
                           PrintOnePageCB, n_pages);
}
```

Examples of **XmNdefaultPixmapResolution** usage:

- An application reuses the same image sources it uses for the video interface, in XBM or XPM, to layout on its printed pages. In this case, scaling is seamless.

```
! icon.xpm is 30x30 pixels
    app*dialog.pushb.labelPixmap:icon.xpm
    ! print is 400dpi
    app.print*form.lab.labelPixmap:icon.xpm
    ! 120x120 pixels on the paper (auto scaling)
```

- An application provides a new set of image files, for a given printer resolution (say 300). It doesn't want automatic scaling by the toolkit for that resolution, it wants scaling based on these 300dpi images for higher resolution. It creates its print shell inside using the name "printHiRes" and adds the following in its resource file:

```
app.printHiRes.defaultPixmapResolution:300
    ! icon300.xpm is 120x120 pixels
    app.printHiRes*form.lab.labelPixmap:icon300.xpm
    ! 120x120 pixels on the paper (no scaling)
```

This way a printer resolution of 600 will result in a scale of a 300 dpi image by 2 (dpi=600 divided by base=300), while a printer resolution of 150 (using default print shell name "print") will use the 100 dpi icon scaled by 1.5 (dpi=150 divided by default base=100).

Related Information

XmPrintSetup(3), **XmRedisplayWidget**(3), **XmPrintToFile**(3), **XmPrintPopupPDM**(3)

XmPrintToFile

Purpose

Retrieves and saves data that would normally be printed by the X Print Server.

Synopsis

#include <Xm/Print.h>

XtEnumXmPrintToFile(
Display*dpy***,**
String*filename***,**
XPFinishProc*finish_proc***,**
XtPointer*client_data***);**

Description

XmPrintToFile hides the details of X display connection and **XpGetDocumentData** to the Motif application programmer.

This function is a convenience routine that hides the details of the X and Xp internals to the application programmer by calling the **XpGetDocumentData** function with appropriate save and finish callbacks.

This is used in the context of X Printing when the user has specified the "print-to-file" option from a regular Print Setup Dialog box.

XmPrintToFile first tries to open the given filename for writing and returns **False** if it can't. Else, it uses **XpGetDocumentData**, giving it a save proc that writes the data received in the file and a finish proc that closes the file or removes it on an unsuccessful termination. It calls **finish_proc** at that point, passing it the argument received from the Xp layer (**status == XPGetDocFinished** means the file is valid and was closed, otherwise the file was removed).

XmPrintToFile is non-blocking; if it returns successfully, it just means the file was opened successfully, not that all the data was received.

dpy	Print display connection.
filename	Name of the file to put the print data in.

finish_proc	Called when all the data has been received.
client_data	Passed with the *finish_proc*.

Return Values

Returns **False** if the filename could not be created or opened for writing, **True** otherwise.

Errors/Warnings

Not applicable

Examples

A typical OK callback from a **DtPrintSetupBox**:

```
PrintOKCallback(widget...)
/*-------------*/
{   int save_data = XPSpool;

    pshell = XmPrintSetup (widget, pbs->print_screen,
                                   "Print", NULL, 0);

    XtAddCallback(pshell, XmNstartJobCallback, startJobCB, data);

    if (pbs->destination == DtPRINT_TO_FILE)
                 save_data = XPGetData;

    /* start job must precede XpGetDocumentData in XmPrintToFile */
    XpStartJob(XtDisplay(pshell), save_data);
    XFlush(XtDisplay(pshell));  /* maintain the sequence
                              between startjob and getdocument */

    /* setup print to file */
    if (pbs->destination == DtPRINT_TO_FILE)
        XmPrintToFile(XtDisplay(pshell),
                              pbs->dest_info, FinishPrintToFile, NULL);
    }

}

static void
startJobCB(Widget, XtPointer call_data, XtPointer client_data)
{
  print(p);   /* rendering happens here */

  XpEndJob(XtDisplay(p->print_shell));

  /* clean up */
  XtDestroyWidget(p->print_shell);
          XtCloseDisplay(XtDisplay(p->print_shell));
}
```

Related Information

XmPrintSetup(3), **XmPrintShell**(3), **XmRedisplayWidget**(3), **XmPrintPopupPDM**(3)

XmProcessTraversal

Purpose

A function that determines which component receives keyboard events when a widget has the focus

Synopsis

#include <Xm/Xm.h>

Boolean XmProcessTraversal(
 Widget *widget***,**
 XmTraversalDirection *direction***);**

Description

XmProcessTraversal determines which component of a hierarchy receives keyboard events when the hierarchy that contains the given widget has keyboard focus.

XmProcessTraversal changes focus only when the keyboard focus policy of the widget hierarchy is explicit. If the **XmNkeyboardFocusPolicy** of the nearest shell ancestor of the given widget is not **XmEXPLICIT**, **XmProcessTraversal** returns False without making any focus changes.

widget	Specifies the widget ID of the widget whose hierarchy is to be traversed
direction	Specifies the direction of traversal

DEFINITIONS

In order to be eligible to receive keyboard focus when the shell's **XmNkeyboardFocusPolicy** is **XmEXPLICIT**, a widget or gadget must meet the following conditions:

- The widget and its ancestors are not in the process of being destroyed.
- The widget and its ancestors are *sensitive*. A widget is sensitive when its **XmNsensitive** and **XmNancestorSensitive** resources are both True.
- The **XmNtraversalOn** resource for the widget and its ancestors is True.

- The widget is viewable. This means that the widget and its ancestors are managed, realized, and (except for gadgets) mapped. Furthermore, in general, some part of the widget's rectangular area must be unobscured by the widget's ancestors. If an application unmaps a widget that has its **XmNmappedWhenManaged** resource set to True, the result is undefined.

  In a ScrolledWindow with an **XmNscrollingPolicy** of **XmAUTOMATIC**, a widget that is obscured because it is not within the clip window may be able to receive focus if some part of the widget is within the work area and if an **XmNtraverseObscuredCallback** routine can make the widget at least partially visible by scrolling the window.

In general only primitives, gadgets, and Drawing Area are eligible to receive focus. Most managers cannot receive focus even if they meet all these conditions.

The *direction* argument identifies the kind of traversal action to take. The descriptions of these actions below refer to traversable non-tab-group widgets and traversable tab groups.

- A traversable non-tab-group widget is a widget that is not a tab group and that meets all the conditions for receiving focus described above.
- A traversable tab group widget is a tab group widget that meets the same conditions, except that a manager that is a tab group and meets the other conditions is also eligible for traversal as long as it contains a descendant that can receive focus.

A tab group is a widget whose **XmNnavigationType** is:

- **XmTAB_GROUP** or **XmSTICKY_TAB_GROUP**, if the hierarchy (up to the nearest shell ancestor) that contains the widget has no widget whose **XmNnavigationType** is **XmEXCLUSIVE_TAB_GROUP**
- **XmEXCLUSIVE_TAB_GROUP** or **XmSTICKY_TAB_GROUP**, if the hierarchy (up to the nearest shell ancestor) that contains the widget has any widget whose **XmNnavigationType** is **XmEXCLUSIVE_TAB_GROUP**

Traversal Actions

The hierarchy to be traversed is that containing the *widget* argument. This hierarchy is traversed only up to the nearest shell; **XmProcessTraversal** does not move focus from one shell to another. If the shell containing *widget* does not currently have the focus, any change that **XmProcessTraversal** makes to the element with focus within that shell does not take effect until the next time the shell receives focus.

XmProcessTraversal begins the traversal action from the widget in the hierarchy that currently has keyboard focus or that last had focus when the user traversed away from the shell hierarchy.

The value of the *direction* argument determines which of three kinds of traversal action to take:

- Traversal to a non-tab-group widget. This kind of traversal is possible only when the widget that currently has focus is not a tab group; otherwise, **XmProcessTraversal** returns False for these actions.

  These actions do not move focus from one tab group to another. The actions first determine the containing tab group. This is the tab group containing the widget that currently has focus. The actions traverse only to a non-tab-group widget within the containing tab group.

  A non-tab-group widget is eligible for this kind of traversal if the widget is traversable and has no tab group ancestors up to the containing tab group. If the tab group contains no traversable non-tab-group widgets, **XmProcessTraversal** returns False.

  Following are the possible values of the *direction* argument. Note that when actions wrap, wrapping occurs in the traversal direction. The following describes what happens in a left to right environment:

  — **XmTRAVERSE_RIGHT**—If the **XmNnavigationType** of the containing tab group is not **XmEXCLUSIVE_TAB_GROUP**, focus moves to the next traversable non-tab-group widget to the right of the widget that currently has focus. In a left to right environment, at the right side of the tab group this action wraps to the non-tab-group widget at the left side and next toward the bottom. At the rightmost widget in the bottom row of the tab group this action wraps to the non-tab-group widget at the leftmost widget in the upper row.

  In a right to left environment, at the right side of the tab group, this action wraps to the non-tab-group widget at the left side and next toward the top. At the rightmost widget in the upper row of the tab group this action wraps to the non-tab-group widget at the leftmost widget in the bottom row.

  If the **XmNnavigationType** of the containing tab group is **XmEXCLUSIVE_TAB_GROUP**, focus moves to the next traversable non-tab-group widget in the tab group, proceeding in the order in which the widgets appear in their parents' **XmNchildren** lists. After the last widget in the tab group, this action wraps to the first non-tab-group widget.

— **XmTRAVERSE_LEFT**—If the **XmNnavigationType** of the containing tab group is not **XmEXCLUSIVE_TAB_GROUP**, focus moves to the next traversable non-tab-group widget to the left of the widget that currently has focus. In a left to right environment, at the left side of the tab group this action wraps to the non-tab-group widget at the right side and next toward the top. At the leftmost widget in the upper row of the tab group this action wraps to the non-tab-group widget at the rightmost widget in the bottom row.

In a right to left environment, at the left side of the tab group this action wraps to the non-tab-group widget at the right side and next toward the bottom. At the leftmost widget in the bottom row of the tab group this action wraps to the non-tab-group widget at the rightmost widget in the upper row.

If the **XmNnavigationType** of the containing tab group is **XmEXCLUSIVE_TAB_GROUP**, focus moves to the previous traversable non-tab-group widget in the tab group, proceeding in the reverse order in which the widgets appear in their parents' **XmNchildren** lists. After the first widget in the tab group, this action wraps to the last non-tab-group widget.

— **XmTRAVERSE_DOWN**—If the **XmNnavigationType** of the containing tab group is not **XmEXCLUSIVE_TAB_GROUP**, focus moves to the next traversable non-tab-group widget below the widget that currently has focus. In a left to right environment, at the bottom of the tab group this action wraps to the non-tab-group widget at the top and next toward the right. At the bottom widget in the rightmost column of the tab group this action wraps to the non-tab-group widget at the top widget in the leftmost column.

In a right to left environment, at the bottom of the tab group this action wraps to the non-tab-group widget at the top and next toward the left. At the bottom widget of the leftmost widget of the tab group this action wraps to the non-tab-group widget at the top widget of the rightmost column.

If the **XmNnavigationType** of the containing tab group is **XmEXCLUSIVE_TAB_GROUP**, focus moves to the next traversable non-tab-group widget in the tab group, proceeding in the order in which the widgets appear in their parents' **XmNchildren** lists. After the last widget in the tab group, this action wraps to the first non-tab-group widget.

— **XmTRAVERSE_UP**—If the **XmNnavigationType** of the containing tab group is not **XmEXCLUSIVE_TAB_GROUP**, focus moves to the next traversable non-tab-group widget above the widget that currently has focus. In a left to right environment, at the top of the tab group this action wraps to the non-tab-group widget at the bottom and next toward the left. At the top widget of the leftmost column of the tab group this action wraps to the non-tab-group widget at the bottom widget of the rightmost column.

In a right to left environment, at the top of the tab group this action wraps to the non-tab-group widget at the bottom and next toward the right. At the top widget of the right most column of the tab group this action wraps to the non-tab-group widget at the bottom widget of the leftmost column.

If the **XmNnavigationType** of the containing tab group is **XmEXCLUSIVE_TAB_GROUP**, focus moves to the previous traversable non-tab-group widget in the tab group, proceeding in the reverse order in which the widgets appear in their parents' **XmNchildren** lists. After the first widget in the tab group, this action wraps to the last non-tab-group widget.

— **XmTRAVERSE_NEXT**—Focus moves to the next traversable non-tab-group widget in the tab group, proceeding in the order in which the widgets appear in their parents' **XmNchildren** lists. After the last widget in the tab group, this action wraps to the first non-tab-group widget.

— **XmTRAVERSE_PREV**—Focus moves to the previous traversable non-tab-group widget in the tab group, proceeding in the reverse order in which the widgets appear in their parents' **XmNchildren** lists. After the first widget in the tab group, this action wraps to the last non-tab-group widget.

— **XmTRAVERSE_HOME**—If the **XmNnavigationType** of the containing tab group is not **XmEXCLUSIVE_TAB_GROUP**, focus moves to the first traversable non-tab-group widget at the initial focus of the tab group.

If the **XmNnavigationType** of the containing tab group is **XmEXCLUSIVE_TAB_GROUP**, focus moves to the first traversable non-tab-group widget in the tab group, according to the order in which the widgets appear in their parents' **XmNchildren** lists.

• Traversal to a tab group. These actions first determine the current widget hierarchy and the containing tab group. The current widget hierarchy is the widget hierarchy whose root is the nearest shell ancestor of the widget that currently has focus. The containing tab group is is the tab group containing the widget that currently has focus. If the current widget hierarchy contains no traversable tab groups, **XmProcessTraversal** returns False.

Following are the possible values of the *direction* argument. If any tab group in the current widget hierarchy has an **XmNnavigationType** of **XmEXCLUSIVE_TAB_GROUP**, traversal of tab groups in the hierarchy proceeds to widgets in the order in which their **XmNnavigationType** resources were specified as **XmEXCLUSIVE_TAB_GROUP** or **XmSTICKY_TAB_GROUP**.:

— **XmTRAVERSE_NEXT_TAB_GROUP**—Finds the hierarchy that contains *widget*, finds the active tab group (if any), and makes the next tab group the active tab group in the hierarchy.

— **XmTRAVERSE_PREV_TAB_GROUP**—Finds the hierarchy that contains *widget*, finds the active tab group (if any), and makes the previous tab group the active tab group in the hierarchy.

- Traversal to any widget. In this case the *widget* argument is the widget to which **XmProcessTraversal** tries to give focus. If the widget is not traversable, **XmProcessTraversal** returns False.

  Following are the possible values of the *direction* argument:

  — **XmTRAVERSE_CURRENT**—Finds the hierarchy and the tab group that contain *widget*. If this tab group is not the active tab group, this action makes it the active tab group. If *widget* is an item in the active tab group, this action makes it the active item. If *widget* is the active tab group, this action makes the first traversable item in the tab group the active item.

CAUTIONS

Using **XmProcessTraversal** to traverse to MenuBars, Pulldown menu panes, or Popup menu panes is not supported.

XmProcessTraversal cannot be called recursively. In particular, an application cannot call this routine from an **XmNfocusCallback** or **XmNlosingFocusCallback** procedure.

Return Values

Returns True if the traversal action succeeded. Returns False if the **XmNkeyboardFocusPolicy** of the nearest shell ancestor of *widget* is not **XmEXPLICIT**, if the traversal action finds no traversable widget to receive focus, or if the call to the routine has invalid arguments.

Related Information

XmGetVisibility(3) and **XmIsTraversable**(3).

XmRedisplayWidget

Purpose

Synchronously activates the **expose** method of a widget to draw its content

Synopsis

#include <Xm/Xm.h>

voidXmRedisplayWidget(
Widget*widget***);**

Description

This function is a convenience routine that hides the details of the Xt internals to the application programmer by calling the **expose** method of the given widget with a well formed **Expose** event and **Region** corresponding to the total area of the widget. If the widget doesn't have an **Expose** method, the function does nothing.

This is primarily used in the context of X Printing if the programming model chosen by the application is *synchronous*; that is, it doesn't rely of X Print events for the driving of page layout but wants to completely control the sequence of rendering requests.

XmRedisplayWidget doesn't clear the widget window prior to calling the **expose** method, since this is handled by calls to **XpStartPage** .

widget	The widget to redisplay.

Return Values

None.

Errors/Warnings

Not applicable

Examples

In the following, a simple application wants to print the content of a multi-page text widget (similar to **dtpad**).

```
PrintOKCallback(print_dialog...)
/*-------------*/
{
    pshell = XmPrintSetup (print_dialog, pbs->print_screen,
                                   "Print", NULL, 0);

    XpStartJob(XtDisplay(pshell), XPSpool);

    /**** here I realize the shell, get its size, create my widget
     hierarchy: a bulletin board, and then a text widget,
     that I stuff with the video text widget buffer */

    /* get the total number of pages to print */
    XtVaGetValues(ptext, XmNrows, &prows,
                         XmNtotalLines, n_lines, NULL);
    n_pages = n_lines / prows;

    /***** now print the pages in a loop */

    for (cur_page=0; cur_page != n_pages; cur_page++) {

               XpStartPage(XtDisplay(pshell), XtWindow(pshell), False);
               XmRedisplayWidget(ptext);  /* do the drawing */
               XpEndPage(XtDisplay(pshell));

        XmTextScroll(ptext, prows);  /* get ready for next page */
    }

    /***** I'm done */
    XpEndJob(XtDisplay(pshell));

}
```

Of course, one could change the above code to include it in a **fork()** branch so that the main program is not blocked while printing is going on. Another way to achieve a "print-in-the-background" effect is to use an Xt workproc. Using the same sample application, that gives us:

```
Boolean
PrintOnePageWP(XtPointer npages) /* workproc */
/*-------------*/
{
    static int cur_page = 0;
    cur_page++;

    XpStartPage(XtDisplay(pshell), XtWindow(pshell), False);
    XmRedisplayWidget(ptext);  /* do the drawing */
    XpEndPage(XtDisplay(pshell));

    XmTextScroll(ptext, prows);  /*  get ready for next page */

    if (cur_page == n_pages) { /***** I’m done */
        XpEndJob(XtDisplay(pshell));

        XtDestroyWidget(pshell);
        XtCloseDisplay(XtDisplay(pshell));
    }

    return (cur_page == n_pages);
}

PrintOKCallback(...)
/*-------------*/
{
    pshell = XmPrintSetup (widget, pbs->print_screen,
                                  "Print", NULL, 0);

    XpStartJob(XtDisplay(pshell), XPSpool);

    /**** here I get the size of the shell, create my widget
          hierarchy: a bulletin board, and then a text widget,
                  that I stuff with the video text widget buffer */

    /* get the total number of pages to print */
    /* ... same code as above example */
    /***** print the pages in the background */
    XtAppAddWorkProc(app_context, PrintOnePageWP, n_pages);
}
```

Related Information

XmPrintSetup(3), **XmPrintShell**(3)

XmRegisterSegmentEncoding

Purpose

A compound string function that registers a compound text encoding format for a specified font list element tag

Synopsis

#include <Xm/Xm.h>

char * XmRegisterSegmentEncoding(
    char ***_fontlist_tag_,**
    char ***_ct_encoding_);**

Description

XmRegisterSegmentEncoding registers a compound text encoding format with the specified font list element tag. The **XmCvtXmStringToCT** function uses this registry to map the font list tags of compound string segments to compound text encoding formats. Registering a font list tag that already exists in the registry overwrites the original entry. You can unregister a font list tag by passing a NULL value for the *ct_encoding* parameter.

fontlist_tag Specifies the font list element tag to be registered. The tag must be a NULL-terminated ISO8859-1 string.

ct_encoding Specifies the compound text character set to be used for segments with the font list tag. The value must be a NULL-terminated ISO8859-1 string. A value of **XmFONTLIST_DEFAULT_TAG** maps the specified font list tag to the code set of the locale.

Return Values

Returns NULL for a new font list tag or the old *ct_encoding* value for an already registered font list tag. The application is responsible for freeing the storage associated with the returned data (if any) by calling **XtFree**.

Related Information

XmCvtXmStringToCT(3), **XmFontList**(3), **XmMapSegmentEncoding**(3), and **XmString**(3).

XmRemoveFromPostFromList

Purpose

a RowColumn function that disables a menu for a particular widget

Synopsis

#include <Xm/RowColumn.h>

void XmRemoveFromPostFromList(
    Widget *menu***,**
    Widget *post_from_widget***);**

Description

XmRemoveFromPostFromList makes a Popup or Pulldown menu no longer accessible from a widget. This function does not destroy a menu, or deallocate the memory associated with it. It simply removes the widget from the menu's list of widgets permitted to post that menu.

If the *menu* argument refers to a Popup menu, the event handlers are removed from the *post_from_widget* widget. If the argument refers to a Pulldown menu, its ID is removed from the **XmNsubMenuId** of the specified *post_from_widget*. Also, if the menu is a Pulldown menu, the *post_from_widget* widget must be either a CascadeButton or a CascadeButtonGadget.

menu Specifies the widget ID of a the Popup or Pulldown menu to be made inaccessible from the *post_from_widget* widget.

post_from_widget
Specifies the widget ID of the widget which can no longer post the menu referred to by the *menu* argument..

For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Related Information

XmAddToPostFromList(3), **XmGetPostedFromWidget**(3), and **XmRowColumn**(3).

XmRemoveProtocolCallback

Purpose

A VendorShell function that removes a callback from the internal list

Synopsis

#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmRemoveProtocolCallback(
    Widget *shell***,**
    Atom *property***,**
    Atom *protocol***,**
    XtCallbackProc *callback***,**
    XtPointer *closure***);**

Description

XmRemoveProtocolCallback removes a callback from the internal list.

XmRemoveWMProtocolCallback is a convenience interface. It calls **XmRemoveProtocolCallback** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocol	Specifies the protocol atom
callback	Specifies the procedure to call when a protocol message is received
closure	Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddProtocolCallback**(3), **XmInternAtom**(3), and **XmRemoveWMProtocolCallback**(3).

XmRemoveProtocols

Purpose

A VendorShell function that removes the protocols from the protocol manager and deallocates the internal tables

Synopsis

#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmRemoveProtocols(
Widget *shell***,**
Atom *property***,**
Atom * *protocols***,**
Cardinal *num_protocols***);**

Description

XmRemoveProtocols removes the protocols from the protocol manager and deallocates the internal tables. If any of the protocols are active, it will update the handlers and update the property if *shell* is realized.

XmRemoveWMProtocols is a convenience interface. It calls **XmRemoveProtocols** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocols	Specifies the protocol atoms
num_protocols	Specifies the number of elements in protocols

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddProtocols**(3), **XmInternAtom**(3), and **XmRemoveWMProtocols**(3).

XmRemoveTabGroup

Purpose

A function that removes a tab group

Synopsis

#include <Xm/Xm.h>

void XmRemoveTabGroup(
Widget *tab_group***);**

Description

This function is obsolete and its behavior is replaced by setting **XmNnavigationType** to **XmNONE**. **XmRemoveTabGroup** removes a widget from the list of tab groups associated with a particular widget hierarchy and sets the widget's **XmNnavigationType** to **XmNONE**.

tab_group Specifies the widget ID

Related Information

XmAddTabGroup(3), **XmManager**(3), and **XmPrimitive**(3).

XmRemoveWMProtocolCallback

Purpose

A VendorShell convenience interface that removes a callback from the internal list

Synopsis

#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmRemoveWMProtocolCallback(
    Widget *shell***,**
    Atom *protocol***,**
    XtCallbackProc *callback***,**
    XtPointer *closure***);**

Description

XmRemoveWMProtocolCallback is a convenience interface. It calls **XmRemoveProtocolCallback** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
protocol	Specifies the protocol atom
callback	Specifies the procedure to call when a protocol message is received
closure	Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddWMProtocolCallback**(3), **XmInternAtom**(3), and **XmRemoveProtocolCallback**(3).

XmRemoveWMProtocols

Purpose

A VendorShell convenience interface that removes the protocols from the protocol manager and deallocates the internal tables

Synopsis

#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmRemoveWMProtocols(
  Widget *shell***,**
  Atom * *protocols***,**
  Cardinal *num_protocols***);**

Description

XmRemoveWMProtocols is a convenience interface. It calls **XmRemoveProtocols** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
protocols	Specifies the protocol atoms
num_protocols	Specifies the number of elements in protocols

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmAddWMProtocols**(3), **XmInternAtom**(3), and **XmRemoveProtocols**(3).

XmRenderTableAddRenditions

Purpose

Creates a new render table

Synopsis

#include <Xm/Xm.h>

XmRenderTable XmRenderTableAddRenditions(
        XmRenderTable *oldtable***,**
        XmRendition **renditions***,**
        Cardinal *rendition_count***,**
        XmMergeMode *merge_mode***);**

Description

XmRenderTableAddRenditions is a function to create a new render table that includes the renditions listed in *oldtable*, if there is one. This function also copies specified renditions (*renditions*) to the new render table. The first *rendition_count* renditions of the *renditions* array are added to the new table. If a rendition is tagged with a tag that matches a tag already in *oldtable*, then the existing rendition using that tag is either modified or freed and replaced with the new rendition, depending on the value of *merge_mode*. If *oldtable* is NULL, **XmRenderTableAddRenditions** creates a new render table containing only the specified renditions.

This function deallocates the original render table after extracting the required information. It is the responsibility of the caller to free the renditions of the *renditions* array by calling the **XmRenditionFree** function.

oldtable Specifies the render table to be added to.

renditions Specifies an array of renditions to be added.

rendition_count
Specifies the number of renditions from *renditions* to be added.

merge_mode Specifies what to do if the **XmNtag** of a rendition matches that of one that already exists in *oldtable*. The possible values are as follows:

XmMERGE_REPLACE
: Completely replaces the old rendition with the new one.

XmMERGE_OLD
: Replaces any unspecified values of the old rendition with the corresponding values from the new rendition.

XmMERGE_NEW
: Replaces the old rendition with the new rendition, replacing any unspecified values of the new rendition with the corresponding values from the old rendition.

XmSKIP
: Skips over the new rendition, leaving the old rendition intact.

Return Values

If *renditions* is NULL or *rendition_count* is 0 (zero), this function returns *oldtable*. Otherwise, the function returns a new **XmRenderTable**. The function allocates space to hold this new render table. The application is responsible for managing this allocated space. The application can recover the allocated space by calling **XmRenderTableFree**.

Related Information

XmRendition(3) and **XmRenderTableFree**(3).

XmRenderTableCopy

Purpose

A render table function that copies renditions

Synopsis

#include <Xm/Xm.h>

XmRenderTable XmRenderTableCopy(
    XmRenderTable *table***,**
    XmStringTag **tags***,**
    int *tag_count***);**

Description

XmRenderTableCopy creates a new render table which will contain the renditions of the *table* whose tags match those in *tags*.

table	Specifies the table containing the renditions to be copied.
tags	Specifies an array of tags, whose corresponding renditions are to be copied. NULL indicates that the complete table should be copied.
tag_count	Specifies the number of tags in *tags*.

Return Values

Returns NULL if *table* is NULL. Otherwise, this function returns the new render table. This function allocates space to hold the new render table. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XmRenderTableFree**.

Related Information

XmRendition(3) and **XmRenderTableFree**(3).

XmRenderTableCvtFromProp

Purpose

A render table function that converts from a string representation to a render table

Synopsis

#include <Xm/Xm.h>

XmRenderTable XmRenderTableCvtFromProp(
    Widget *widget*,
    **char ****property*,
    unsigned int *length*);

Description

XmRenderTableCvtFromProp converts a string of characters representing a render table to a render table. This routine is typically used by the destination of a data transfer operation to produce a render table from a transferred representation.

widget	Specifies the widget that is the destination for the render table
property	Specifies a string of characters representing a render table
length	Specifies the number of bytes in *property*

Return Values

Returns a render table. The function allocates space to hold the returned render table. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XmRenderTableFree**.

Related Information

XmRenderTable(3), **XmRenderTableCvtToProp**(3), and **XmRenderTableFree**(3).

XmRenderTableCvtToProp

Purpose

A render table function that converts a render table to a string representation

Synopsis

#include <Xm/Xm.h>

unsigned int XmRenderTableCvtToProp(
 Widget *widget***,**
 XmRenderTable *table***,**
 char *****prop_return*);**

Description

XmRenderTableCvtToProp converts a render table to a string of characters representing the render table. This routine is typically used by the source of a data transfer operation to produce a representation for transferring a render table to a destination.

widget	Specifies the widget that is the source of the render table
table	Specifies a render table to be converted
prop_return	Specifies a pointer to a string that is created and returned by this function. The function allocates space to hold the returned string. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XtFree**.

Return Values

Returns the number of bytes in the string representation.

Related Information

XmRenderTable(3) and **XmRenderTableCvtFromProp**(3).

XmRenderTableFree

Purpose

A render table function that recovers memory

Synopsis

#include <Xm/Xm.h>

void XmRenderTableFree(
    XmRenderTable *table***);**

Description

XmRenderTableFree frees the memory associated with the specified render *table*.

table	Specifies the table to be freed.

Related Information

XmRendition(3).

XmRenderTableGetRendition

Purpose

A convenience function that matches a rendition tag

Synopsis

#include <Xm/Xm.h>

XmRendition XmRenderTableGetRendition(
    XmRenderTable *table***,**
    XmStringTag *tag***);**

Description

XmRenderTableGetRendition searches *table* and returns a copy of the rendition whose **XmNtag** resource matches *tag*. If no rendition matches, then NULL is returned. This function is to be used for just one rendition match.

It is the responsibility of the caller to free the returned rendition with the **XmRenditionFree** function.

table	Specifies the table containing renditions to be searched.
tag	Specifies the tag to search for.

Return Values

Returns NULL if there is no match; otherwise, this function returns a new **XmRendition**.

Related Information

XmRenderTableGetRenditions(3), **XmRenderTableGetTags**(3), and **XmRendition**(3).

XmRenderTableGetRenditions

Purpose

A convenience function that matches rendition tags

Synopsis

#include <Xm/Xm.h>

XmRendition *XmRenderTableGetRenditions(
  XmRenderTable *table***,**
  XmStringTag **tags***,**
  Cardinal *tag_count***);**

Description

XmRenderTableGetRenditions searches *table* and returns an array of copies of the renditions whose **XmNtag** resources match a tag in *tags*. If no renditions match, then NULL is returned. The size of the returned array is *tag_count*. The **XmNtag** resource of each rendition will match the corresponding tag in *tags*. If no match is found for a particular tag, the corresponding slot in the return value will be NULL.

It is the responsibility of the caller to call the **XmRenditionFree** function to free the new renditions, and the **XtFree** function to free the array.

table	Specifies the table containing renditions to be searched.
tags	Specifies the tags to search for.
tag_count	Specifies the number of tags in *tags*.

Return Values

Returns NULL if there is no match; otherwise, this function returns an array of new **XmRendition**s.

Related Information

XmRenderTableGetRendition(3), **XmRenderTableGetTags**(3), and **XmRendition**(3).

XmRenderTableGetTags

Purpose

A convenience function that gets rendition tags

Synopsis

#include <Xm/Xm.h>

int XmRenderTableGetTags(
 XmRenderTable *table***,**
 XmStringTag ***tag_list***);**

Description

XmRenderTableGetTags searches the specified *table* for the **XmNtag** resources of all the renditions (**XmRendition**s) entries. These tag resources are then composed into an array.

table	Specifies the table containing the **XmRendition**s.
tag_list	Is the array of *XmStringTags* generated by this function. The function allocates space to hold the returned tags and to hold the *tag_list* itself. The application is responsible for managing this allocated space. This application can recover this allocated space by calling **XtFree** once for each of the returned tags, and then calling **XtFree** on the returned *tag_list* variable itself.

Return Values

Returns the number of tags in *tag_list*.

Related Information

XmRendition(3).

XmRenderTableRemoveRenditions

Purpose

A convenience function that removes renditions

Synopsis

#include <Xm/Xm.h>

XmRenderTable XmRenderTableRemoveRenditions(
    XmRenderTable *oldtable***,**
    XmStringTag **tags***,**
    int *tag_count***);**

Description

XmRenderTableRemoveRenditions removes from *oldtable* the renditions whose tags match the tags specified in *tags*, then places the remaining renditions in a newly created render table.

oldtable	Specifies the render table from which renditions are to be removed. This function deallocates the original render table and the matching renditions after extracting the required information.
tags	Specifies an array of tags, whose corresponding renditions are to be removed from *oldtable*.
tag_count	Specifies the number of tags in *tags*.

Return Values

If *oldtable* or *tags* is NULL, or *tag_count* is 0 (zero), or no renditions are removed from *oldtable*, this function returns *oldtable*. Otherwise, it returns a newly allocated **XmRenderTable**. The application is responsible for managing this allocated render table. The application can recover this allocated space by calling **XmRenderTableFree**.

Related Information

XmRendition(3) and **XmRenderTableFree**(3).

XmRenditionCreate

Purpose

A convenience function that creates a rendition

Synopsis

#include <Xm/Xm.h>

XmRendition XmRenditionCreate(
    Widget *widget***,**
    XmStringTag *tag***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmRenditionCreate creates a rendition whose resources are set to the values specified in *arglist*. Default values are assigned to resources that are not specified.

widget	Specifies the widget used for deriving any necessary information for creating the rendition. In particular, the X display of *widget* will be used for loading fonts.
tag	Specifies the tag for the rendition. (This will become the **XmNtag** resource for the rendition.)
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

Return Values

Returns the created rendition. The function allocates space to hold the returned rendition. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XmRenditionFree**.

Related Information

XmRendition(3) and **XmRenditionFree**(3).

XmRenditionFree

Purpose

A convenience function that frees a rendition

Synopsis

```
#include <Xm/Xm.h>

void XmRenditionFree(
        XmRendition rendition);
```

Description

XmRenditionFree recovers memory used by *rendition*.

rendition Specifies the rendition to be freed.

Related Information

XmRendition(3).

XmRenditionRetrieve

Purpose

A convenience function that retrieves rendition resources

Synopsis

#include <Xm/Xm.h>

void XmRenditionRetrieve(
    XmRendition *rendition***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmRenditionRetrieve extracts values for the given resources (*arglist*) from the specified rendition. Note that the function returns the actual values of the resources, not copies. Therefore it is necessary to copy before modifying any resource whose value is an address. This will include such resources as **XmNfontName**, **XmNfont**, and **XmNtabList**.

rendition	Specifies the rendition.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

Related Information

XmRendition(3) and **XmTabListCopy**(3).

XmRenditionUpdate

Purpose

A convenience function that modifies resources

Synopsis

#include <Xm/Xm.h>

void XmRenditionUpdate(
    XmRendition *rendition***,**
    ArgList *arglist***,**
    Cardinal *argcount***);**

Description

XmRenditionUpdate modifies resources in the specified rendition.

rendition	Specifies the rendition.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*).

Related Information

XmRendition(3).

XmRepTypeAddReverse

Purpose

A representation type manager function that installs the reverse converter for a previously registered representation type

Synopsis

#include <Xm/RepType.h>

void XmRepTypeAddReverse(
    XmRepTypeId *rep_type_id***);**

Description

XmRepTypeAddReverse installs the reverse converter for a previously registered representation type. The reverse converter takes a numerical representation type value and returns its corresponding string value. Certain applications may require this capability to obtain a string value to display on a screen or to build a resource file.

The *values* argument of the **XmRepTypeRegister** function can be used to register representation types with nonconsecutive values or with duplicate names for the same value. If the list of numerical values for a representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value. For example, if a *value_names* array has *cancel*, *proceed*, and *abort*, and the corresponding *values* array contains 0, 1, and 0, the reverse converter will return *cancel* instead of *abort* for an input value of 0.

rep_type_id Specifies the identification number of the representation type

Related Information

XmRepTypeGetId(3) and **XmRepTypeRegister**(3).

XmRepTypeGetId

Purpose

A representation type manager function that retrieves the identification number of a representation type

Synopsis

```
#include <Xm/RepType.h>

XmRepTypeId XmRepTypeGetId(
        String rep_type);
```

Description

XmRepTypeGetId searches the registration list for the specified representation type and returns the associated identification number.

rep_type	Specifies the representation type for which an identification number is requested

Return Values

Returns the identification number of the specified representation type. If the representation type is not registered, the function returns **XmREP_TYPE_INVALID**.

Related Information

XmRepTypeGetRegistered(3) and **XmRepTypeRegister**(3).

XmRepTypeGetNameList

Purpose

A representation type manager function that generates a list of values for a representation type

Synopsis

#include <Xm/RepType.h>

String * XmRepTypeGetNameList(
    XmRepTypeId *rep_type_id***,**
    Boolean *use_uppercase_format***);**

Description

XmRepTypeGetNameList generates a NULL-terminated list of the value names associated with the specified representation type. Each value name is a NULL-terminated string. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

rep_type_id Specifies the identification number of the representation type.

use_uppercase_format
Specifies a Boolean value that controls the format of the name list. If the value is True, each value name is in uppercase characters prefixed by **Xm**; if it is False, the names are in lowercase characters.

Return Values

Returns a pointer to an array of the value names.

Related Information

XmRepTypeGetId(3), **XmRepTypeGetRegistered**(3), and **XmRepTypeRegister**(3).

XmRepTypeGetRecord

Purpose

A representation type manager function that returns information about a representation type

Synopsis

#include <Xm/RepType.h>

XmRepTypeEntry XmRepTypeGetRecord(
        XmRepTypeId *rep_type_id***);**

Description

XmRepTypeGetRecord retrieves information about a particular representation type that is registered with the representation type manager. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

rep_type_id	The identification number of the representation type

The representation type entry structure contains the following information:

```
typedef struct
{
      String  rep_type_name;
      String  *value_names;
      unsigned char   *values;
      unsigned char   num_values;
      Boolean reverse_installed;
      XmRepTypeId     rep_type_id;
} XmRepTypeEntryRec, *XmRepTypeEntry;
```

rep_type_name	The name of the representation type
value_names	An array of representation type value names
values	An array of representation type numerical values
num_values	The number of values associated with the representation type

reverse_installed
A flag that indicates whether or not the reverse converter is installed

rep_type_id The identification number of the representation type

Return Values

Returns a pointer to the representation type entry structure that describes the representation type.

Related Information

XmRepTypeGetId(3), **XmRepTypeGetRegistered**(3), and **XmRepTypeRegister**(3).

XmRepTypeGetRegistered

Purpose

A representation type manager function that returns a copy of the registration list

Synopsis

#include <Xm/RepType.h>

XmRepTypeList XmRepTypeGetRegistered(
  void);

Description

XmRepTypeGetRegistered retrieves information about all representation types that are registered with the representation type manager. The registration list is an array of structures, each of which contains information for a representation type entry. The end of the registration list is marked with a representation type entry whose *rep_type_name* field has a NULL pointer. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

The representation type entry structure contains the following information:

```
typedef struct
{
        String  rep_type_name;
        String  *value_names;
        unsigned char   *values;
        unsigned char   num_values;
        Boolean reverse_installed;
        XmRepTypeId     rep_type_id;
} XmRepTypeEntryRec, *XmRepTypeList;
```

rep_type_name
The name of the representation type

value_names An array of representation type value names

values An array of representation type numerical values

num_values The number of values associated with the representation type

reverse_installed
A flag that indicates whether or not the reverse converter is installed

rep_type_id The identification number of the representation type

Return Values

Returns a pointer to the registration list of representation types.

Related Information

XmRepTypeRegister(3) and **XmRepTypeGetRecord**(3).

XmRepTypeInstallTearOffModelConverter

Purpose

A representation type manager function that installs the resource converter for XmNtearOffModel.

Synopsis

#include <Xm/RepType.h>

```
void XmRepTypeInstallTearOffModelConverter(
          void);
```

Description

XmRepTypeInstallTearOffModelConverter installs the resource converter that allows values for the **XmNtearOffModel** resource to be specified in resource default files.

Related Information

XmRowColumn(3).

XmRepTypeRegister

Purpose

A representation type manager function that registers a representation type resource

Synopsis

#include <Xm/RepType.h>

XmRepTypeId XmRepTypeRegister(
    String *rep_type*,
    String **value_names*,
    unsigned char **values*,
    unsigned char *num_values***);**

Description

XmRepTypeRegister registers a representation type resource with the representation type manager. All features of the representation type management facility become available for the specified representation type. The function installs a forward type converter to convert string values to numerical representation type values.

When the *values* argument is NULL, consecutive numerical values are assumed. The order of the strings in the *value_names* array determines the numerical values for the resource. For example, the first value name is 0 (zero); the second value name is 1; and so on.

If it is non-NULL, the *values* argument can be used to assign values to representation types that have nonconsecutive values or have duplicate names for the same value. Representation types registered in this manner will consume additional storage and will be slightly slower than representation types with consecutive values.

A representation type can only be registered once; if the same representation type name is registered more than once, the behavior is undefined.

The function **XmRepTypeAddReverse** installs a reverse converter for a registered representation type. The reverse converter takes a representation type numerical value and returns the corresponding string value. If the list of numerical values for a representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value. For example, if a *value_names* array has *cancel*, *proceed*, and *abort*, and the corresponding *values* array contains 0, 1, and 0, the reverse converter will return *cancel* instead of *abort* for an input value of 0.

rep_type	Specifies the representation type name.
value_names	Specifies a pointer to an array of value names associated with the representation type. A value name is specified in lowercase characters without an **Xm** prefix. Words within a name are separated with underscores.
values	Specifies a pointer to an array of values associated with the representation type. A value in this array is associated with the value name in the corresponding position of the *value_names* array.
num_values	Specifies the number of entries in the *value_names* and *values* arrays.

Return Values

Returns the identification number for the specified representation type.

Related Information

XmRepTypeAddReverse(3), **XmRepTypeGetId**(3), **XmRepTypeGetNameList**(3), **XmRepTypeGetRecord**(3), **XmRepTypeGetRegistered**(3), and **XmRepTypeValidValue**(3).

XmRepTypeValidValue

Purpose

A representation type manager function that tests the validity of a numerical value of a representation type resource

Synopsis

#include <Xm/RepType.h>

Boolean XmRepTypeValidValue(
    XmRepTypeId *rep_type_id***,**
    unsigned char *test_value***,**
    Widget *enable_default_warning***);**

Description

XmRepTypeValidValue tests the validity of a numerical value for a given representation type resource. The function generates a default warning message if the value is invalid and the *enable_default_warning* argument is non-NULL.

rep_type_id Specifies the identification number of the representation type.

test_value Specifies the numerical value to test.

enable_default_warning
Specifies the ID of the widget that contains a default warning message. If this parameter is NULL, no default warning message is generated and the application must provide its own error handling.

Return Values

Returns True if the specified value is valid; otherwise, returns False.

Related Information

XmRepTypeGetId(3) and **XmRepTypeRegister**(3).

XmResolveAllPartOffsets

Purpose

A function that allows writing of upward-compatible applications and widgets

Synopsis

#include <Xm/Xm.h>

void XmResolveAllPartOffsets(
    WidgetClass *widget_class***,**
    XmOffsetPtr * *offset***,**
    XmOffsetPtr * *constraint_offset***);**

Description

Note: This routine is obsolete and exists for compatibility with previous releases. You should call **XmeResolvePartOffsets** instead.

The use of offset records requires two extra global variables per widget class. The variables consist of pointers to arrays of offsets into the widget record and constraint record for each part of the widget structure. The **XmResolveAllPartOffsets** function allocates the offset records needed by an application to guarantee upward-compatible access to widget instance and constraint records by applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the steps described in the following paragraphs.

Instead of creating a resource list, the widget creates an offset resource list. To accomplish this, use the **XmPartResource** structure and the **XmPartOffset** macro. The **XmPartResource** data structure looks just like a resource list, but instead of having one integer for its offset, it has two shorts. This structure is put into the class record as if it were a normal resource list. Instead of using **XtOffset** for the offset, the widget uses **XmPartOffset**.

If the widget is a subclass of the Constraint class and it defines additional constraint resources, create an offset resource list for the constraint part as well. Instead of using **XtOffset** for the offset, the widget uses **XmConstraintPartOffset** in the constraint resource list.

```
XmPartResource resources[] = {
        {       BarNxyz, BarCXyz, XmRBoolean, sizeof(Boolean),
                XmPartOffset(Bar,xyz), XmRImmediate, (XtPointer)False } };
XmPartResource constraints[] = {
        {       BarNmaxWidth, BarNMaxWidth,
          XmRDimension, sizeof(Dimension),
          XmConstraintPartOffset(Bar,max_width),
          XmRImmediate, (XtPointer)100 } };
```

Instead of putting the widget size in the class record, the widget puts the widget part size in the same field. If the widget is a subclass of the Constraint class, instead of putting the widget constraint record size in the class record, the widget puts the widget constraint part size in the same field.

Instead of putting **XtVersion** in the class record, the widget puts **XtVersionDontCheck** in the class record.

Define a variable, of type **XmOffsetPtr**, to point to the offset record. If the widget is a subclass of the Constraint class, define a variable of type **XmOffsetPtr** to point to the constraint offset record. These can be part of the widget's class record or separate global variables.

In class initialization, the widget calls **XmResolveAllPartOffsets**, passing it pointers to the class record, the address of the offset record, and the address of the constraint offset record. If the widget not is a subclass of the Constraint class, it should pass NULL as the address of the constraint offset record. This does several things:

- Adds the superclass (which, by definition, has already been initialized) size field to the part size field
- If the widget is a subclass of the Constraint class, adds the superclass constraint size field to the constraint size field
- Allocates an array based upon the number of superclasses
- If the widget is a subclass of the constraint class, allocates an array for the constraint offset record
- Fills in the offsets of all the widget parts and constraint parts with the appropriate values, determined by examining the size fields of all superclass records
- Uses the part offset array to modify the offset entries in the resource list to be real offsets, in place

The widget defines a constant that will be the index to its part structure in the offsets array. The value should be 1 greater than the index of the widget's superclass. Constants defined for all **Xm** widgets can be found in **XmP.h**.

```
#define BarIndex (XmBulletinBIndex + 1)
```

Instead of accessing fields directly, the widget must always go through the offset table. The **XmField** and **XmConstraintField** macros help you access these fields. Because the **XmPartOffset**, **XmConstraintPartOffset**, **XmField**, and **XmConstraintField** macros concatenate things, you must ensure that there is no space after the part argument. For example, the following macros do not work because of the space after the part (Label) argument:

```
XmField(w, offset, Label, text, char *)
XmPartOffset(Label, text).
```

Therefore, you must not have any spaces after the part (Label) argument, as illustrated here:

```
XmField(w, offset, Label, text, char *)
```

You can define macros for each field to make this easier. Assume an integer field *xyz*:

```
#define BarXyz(w) (*(int *)(((char *) w) + \
        offset[BarIndex] + XtOffset(BarPart,xyz)))
```

For constraint field *max_width*:

```
#define BarMaxWidth(w) \
        XmConstraintField(w,constraint_offsets,Bar,max_width,Dimension)
```

The parameters for **XmResolveAllPartOffsets** are

widget_class Specifies the widget class pointer for the created widget

offset Returns the offset record

constraint_offset
Returns the constraint offset record

Related Information

XmResolvePartOffsets(3).

XmResolvePartOffsets

Purpose

A function that allows writing of upward-compatible applications and widgets

Synopsis

#include <Xm/Xm.h>

void XmResolvePartOffsets(
WidgetClass *widget_class***,**
XmOffsetPtr * *offset***);**

Description

The use of offset records requires one extra global variable per widget class. The variable consists of a pointer to an array of offsets into the widget record for each part of the widget structure. The **XmResolvePartOffsets** function allocates the offset records needed by an application to guarantee upward-compatible access to widget instance records by applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the steps described in the following paragraphs.

Instead of creating a resource list, the widget creates an offset resource list. To accomplish this, use the **XmPartResource** structure and the **XmPartOffset** macro. The **XmPartResource** data structure looks just like a resource list, but instead of having one integer for its offset, it has two shorts. This structure is put into the class record as if it were a normal resource list. Instead of using **XtOffset** for the offset, the widget uses **XmPartOffset**.

```
XmPartResource resources[] = {
  { BarNxyz, BarCXyz, XmRBoolean,
    sizeof(Boolean), XmPartOffset(Bar,xyz),
    XmRImmediate, (XtPointer)False }
};
```

Instead of putting the widget size in the class record, the widget puts the widget part size in the same field.

Instead of putting **XtVersion** in the class record, the widget puts **XtVersionDontCheck** in the class record.

The widget defines a variable, of type **XmOffsetPtr**, to point to the offset record. This can be part of the widget's class record or a separate global variable.

In class initialization, the widget calls **XmResolvePartOffsets**, passing it a pointer to contain the address of the offset record and the class record. This does several things:

- Adds the superclass (which, by definition, has already been initialized) size field to the part size field
- Allocates an array based upon the number of superclasses
- Fills in the offsets of all the widget parts with the appropriate values, determined by examining the size fields of all superclass records
- Uses the part offset array to modify the offset entries in the resource list to be real offsets, in place

The widget defines a constant that will be the index to its part structure in the offsets array. The value should be 1 greater than the index of the widget's superclass. Constants defined for all **Xm** widgets can be found in **XmP.h**.

```
#define BarIndex (XmBulletinBIndex + 1)
```

Instead of accessing fields directly, the widget must always go through the offset table. The **XmField** macro helps you access these fields. Because the **XmPartOffset** and **XmField** macros concatenate things together, you must ensure that there is no space after the part argument. For example, the following macros do not work because of the space after the part (Label) argument:

```
XmField(w, offset, Label, text, char *)
XmPartOffset(Label, text)
```

Therefore, you must not have any spaces after the part (Label) argument, as illustrated here:

```
XmField(w, offset, Label, text, char *)
```

You can define macros for each field to make this easier. Assume an integer field *xyz*:

```
#define BarXyz(w) (*(int *)(((char *) w) + \
        offset[BarIndex] + XtOffset(BarPart,xyz)))
```

The parameters for **XmResolvePartOffsets** are

widget_class	Specifies the widget class pointer for the created widget
offset	Returns the offset record

Related Information

XmResolveAllPartOffsets(3).

XmScaleGetValue

Purpose

A Scale function that returns the current slider position

Synopsis

#include <Xm/Scale.h>

void XmScaleGetValue(
  Widget *widget***,**
  int * *value_return***);**

Description

XmScaleGetValue returns the current slider position value displayed in the scale.

widget	Specifies the Scale widget ID
value_return	Returns the current slider position value

For a complete definition of Scale and its associated resources, see **XmScale**(3).

Related Information

XmScale(3).

XmScaleSetTicks

Purpose

A Scale function that controls tick marks

Synopsis

#include <Xm/Scale.h>

void XmScaleSetTicks(
    Widget *scale***,**
    int *big_every***,**
    Cardinal *num_medium***,**
    Cardinal *num_small***,**
    Dimension *size_big***,**
    Dimension *size_medium***,**
    Dimension *size_small***);**

Description

XmScaleSetTicks controls the number, location, and size of the tick marks on a Scale. Each tick mark is a SeparatorGadget oriented perpendicular to the Scale's orientation. For example, if the Scale is oriented horizontally, the tick marks will be oriented vertically.

If you specify tick marks for a Scale and then change the Scale's orientation, you will have to do the following:

- Remove all the tick marks. To remove tick marks from a Scale, you must destroy (with *XtDestroyChildren*) the SeparatorGadget tick marks. The first two children of a Scale are its title and scroll bar, and all additional children are tick marks.
- Recreate the tick marks by calling **XmScaleSetTicks**.

scale Specifies the Scale widget ID that is getting the tick marks.

big_every Specifies the number of scale values between big ticks.

num_medium
Specifies the number of medium ticks between big values.

num_small	Specifies the number of small ticks between medium values.
size_big	Specifies the size (either width or height) of the big ticks.
size_medium	Specifies the size (either width or height) of the medium ticks.
size_small	Specifies the size (either width or height) of the small ticks.

For a complete definition of Scale and its associated resources, see **XmScale**(3).

Related Information

XmScale(3).

XmScaleSetValue

Purpose

A Scale function that sets a slider value

Synopsis

#include <Xm/Scale.h>

void XmScaleSetValue(
    Widget *widget***,**
    int *value***);**

Description

XmScaleSetValue sets the slider *value* within the Scale widget.

widget	Specifies the Scale widget ID.
value	Specifies the slider position along the scale. This sets the **XmNvalue** resource.

For a complete definition of Scale and its associated resources, see **XmScale**(3).

Related Information

XmScale(3).

XmScrollBarGetValues

Purpose

A ScrollBar function that returns the ScrollBar's increment values

Synopsis

#include <Xm/ScrollBar.h>
void XmScrollBarGetValues (*widget*, *value_return*, *slider_size_return*, *increment_return*, *page_increment_return***)**
    Widget *widget***;**
    int * *value_return***;**
    int * *slider_size_return***;**
    int * *increment_return***;**
    int * *page_increment_return***;**

Description

XmScrollBarGetValues returns the the ScrollBar's increment values. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main ScrollBar or set slider function attributes.

widget Specifies the ScrollBar widget ID.

value_return Returns the ScrollBar's slider position between the **XmNminimum** and **XmNmaximum** resources. Specify NULL to prevent the return of a particular value.

slider_size_return
Returns the size of the slider as a value between 0 (zero) and the absolute value of **XmNmaximum** minus **XmNminimum**. The size of the slider varies, depending on how much of the slider scroll area it represents.

increment_return
Returns the amount of increment and decrement.

page_increment_return
Returns the amount of page increment and decrement.

For a complete definition of ScrollBar and its associated resources, see **XmScrollBar**(3).

Return Values

Returns the ScrollBar's increment values.

Related Information

XmScrollBar(3).

XmScrollBarSetValues

Purpose

A ScrollBar function that changes ScrollBar's increment values and the slider's size and position

Synopsis

#include <Xm/ScrollBar.h>

void XmScrollBarSetValues (*widget, value, slider_size, increment, page_increment, notify***)**

```
        Widget  widget;
        int     value;
        int     slider_size;
        int     increment;
        int     page_increment;
        Boolean notify;
```

Description

XmSetScrollBarValues changes the ScrollBar's increment values and the slider's size and position. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main ScrollBar or set slider function attributes.

widget	Specifies the ScrollBar widget ID.
value	Specifies the ScrollBar's slider position. Refer to the **XmNvalue** resource described on **XmScrollBar**(3).
slider_size	Specifies the size of the slider. Refer to the **XmNsliderSize** resource described on **XmScrollBar**(3). This argument sets that resource. Specify a value of 0 (zero) if you do not want to change the value.
increment	Specifies the amount of button increment and decrement. Refer to the **XmNincrement** resource described on **XmScrollBar**(3). This argument sets that resource. Specify a value of 0 (zero) if you do not want to change the value.

page_increment
: Specifies the amount of page increment and decrement. Refer to the **XmNpageIncrement** resource described on **XmScrollBar**(3). This argument sets that resource. Specify a value of 0 (zero) if you do not want to change the value.

notify
: Specifies a Boolean value that, when True, indicates a change in the ScrollBar value and also specifies that the ScrollBar widget automatically activates the **XmNvalueChangedCallback** with the recent change. If it is set to False, it specifies any change that has occurred in the ScrollBar's value, but does not activate **XmNvalueChangedCallback**.

For a complete definition of ScrollBar and its associated resources, see **XmScrollBar**(3).

Related Information

XmScrollBar(3).

XmScrollVisible

Purpose

A ScrolledWindow function that makes an invisible descendant of a ScrolledWindow work area visible

Synopsis

#include <Xm/ScrolledW.h>

void XmScrollVisible(
  Widget *scrollw_widget***,**
  Widget *widget***,**
  Dimension *left_right_margin***,**
  Dimension *top_bottom_margin***);**

Description

XmScrollVisible makes an obscured or partially obscured widget or gadget descendant of a ScrolledWindow work area visible. The function repositions the work area and sets the specified margins between the widget and the nearest viewport boundary. The widget's location relative to the viewport determines whether one or both of the margins must be adjusted. This function requires that the **XmNscrollingPolicy** of the ScrolledWindow widget be set to **XmAUTOMATIC**.

scrollw_widget
: Specifies the ID of the ScrolledWindow widget whose work area window contains an obscured descendant.

widget
: Specifies the ID of the widget to be made visible.

left_right_margin
: Specifies the margin to establish between the left or right edge of the widget and the associated edge of the viewport. This margin is established only if the widget must be moved horizontally to make it visible.

top_bottom_margin
: Specifies the margin to establish between the top or bottom edge of the widget and the associated edge of the viewport. This margin is

established only if the widget must be moved vertically to make it visible.

For a complete definition of ScrolledWindow and its associated resources, see **XmScrolledWindow**(3)

Related Information

XmScrolledWindow(3).

XmScrolledWindowSetAreas

Purpose

A ScrolledWindow function that adds or changes a window work region and a horizontal or vertical ScrollBar widget to the ScrolledWindow widget

Synopsis

#include <Xm/ScrolledW.h>

void XmScrolledWindowSetAreas(
    Widget *widget***,**
    Widget *horizontal_scrollbar***,**
    Widget *vertical_scrollbar***,**
    Widget *work_region***);**

Description

XmScrolledWindowSetAreas adds or changes a window work region and a horizontal or vertical ScrollBar widget to the ScrolledWindow widget for the application. Each widget is optional and may be passed as NULL. This function is obsolete and exists for compatibility with other releases. Use the **XmNscrolledWindowChildType** resource of **XmScrolledWindow** instead.

widget Specifies the ScrolledWindow widget ID.

horizontal_scrollbar
Specifies the ScrollBar widget ID for the horizontal ScrollBar to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The resource name associated with this argument is **XmNhorizontalScrollBar**.

vertical_scrollbar
Specifies the ScrollBar widget ID for the vertical ScrollBar to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The resource name associated with this argument is **XmNverticalScrollBar**.

work_region Specifies the widget ID for the work window to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The attribute name associated with this argument is **XmNworkWindow**.

For a complete definition of ScrolledWindow and its associated resources, see **XmScrolledWindow**(3).

Related Information

XmScrolledWindow(3).

XmSelectionBoxGetChild

Purpose

A SelectionBox function that is used to access a component

Synopsis

#include <Xm/SelectioB.h>

Widget XmSelectionBoxGetChild(
    Widget *widget***,**
    unsigned char *child***);**

Description

XmSelectionBoxGetChild is used to access a component within a SelectionBox. The parameters given to the function are the SelectionBox widget and a value indicating which component to access.

NOTE: This routine is obsolete and exists for compatibility with previous releases. Instead of calling **XmSelectionBoxGetChild**, you should call **XtNameToWidget** as described in the **XmSelectionBox**(3) reference page.

widget Specifies the SelectionBox widget ID.

child Specifies a component within the SelectionBox. The following values are legal for this parameter:

- **XmDIALOG_APPLY_BUTTON**
- **XmDIALOG_CANCEL_BUTTON**
- **XmDIALOG_DEFAULT_BUTTON**
- **XmDIALOG_HELP_BUTTON**
- **XmDIALOG_LIST**
- **XmDIALOG_LIST_LABEL**
- **XmDIALOG_OK_BUTTON**
- **XmDIALOG_SELECTION_LABEL**

- **XmDIALOG_SEPARATOR**
- **XmDIALOG_TEXT**
- **XmDIALOG_WORK_AREA**

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox**(3).

Return Values

Returns the widget ID of the specified SelectionBox component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmSelectionBox(3).

XmSetColorCalculation

Purpose

A function to set the procedure used for default color calculation

Synopsis

#include <Xm/Xm.h>

XmColorProc XmSetColorCalculation(
    XmColorProc *color_proc***);**

Description

XmSetColorCalculation sets the procedure to calculate default colors. This procedure is used to calculate the foreground, top shadow, bottom shadow, and select colors on the basis of a given background color. If called with an argument of NULL, it restores the default procedure used to calculate colors.

color_proc Specifies the procedure to use for color calculation.

Following is a description of the **XmColorProc** type used by **XmSetColorCalculation**:

void (**color_proc***) (***background_color, foreground_color, select_color, top_shadow_color, bottom_shadow_color***)**
  XColor **background_color***;**
  XColor **foreground_color***;**
  XColor **select_color***;**
  XColor **top_shadow_color***;**
  XColor **bottom_shadow_color***;**

color_proc Specifies the procedure used to calculate default colors.

The procedure is passed a pointer to an *XColor* structure representing the background color. The *pixel*, *red*, *green*, and *blue* members of this structure are filled in with values that are valid for the current colormap.

The procedure is passed pointers to *XColor* structures representing the foreground, select, top shadow, and bottom shadow colors to be calculated. The procedure calculates and fills in the *red*, *green*, and *blue* members of these structures. The procedure should not allocate color cells for any of these colors.

background_color
Specifies the background color.

foreground_color
Specifies the foreground color to be calculated.

select_color Specifies the select color to be calculated.

top_shadow_color
Specifies the top shadow color to be calculated.

bottom_shadow_color
Specifies the bottom shadow color to be calculated.

Return Values

Returns the color calculation procedure that was used at the time this routine was called.

Related Information

XmChangeColor(3), **XmGetColors**(3), and **XmGetColorCalculation**(3).

XmSetFontUnit

Purpose

A function that sets the font unit value for a display

Synopsis

#include <Xm/Xm.h>

void XmSetFontUnit(
    Display * *display***,**
    int *font_unit_value***);**

Description

XmSetFontUnit provides an external function to initialize font unit values. Applications may want to specify resolution-independent data based on a global font size. See the **XmNunitType** resource description in the reference pages for **XmGadget**, **XmManager**, and **XmPrimitive** for more information on resolution independence.

This function sets the font units for all screens on the display.

NOTE: XmSetFontUnit is obsolete and exists for compatibility with previous releases. Instead of using this function, provide initial values or call **XtSetValues** for the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

display Defines the display for which this font unit value is to be applied.

font_unit_value
Specifies the value to be used for both horizontal and vertical font units in the conversion calculations.

Related Information

XmConvertUnits(3), **XmSetFontUnits**(3), **XmGadget**(3), **XmManager**(3), **XmPrimitive**(3), and **XmScreen**(3).

XmSetFontUnits

Purpose

A function that sets the font unit value for a display

Synopsis

#include <Xm/Xm.h>

void XmSetFontUnits(
Display * *display*,
int *h_value*,
int *v_value*);

Description

XmSetFontUnits provides an external function to initialize font unit values. Applications may want to specify resolution-independent data based on a global font size. This function must be called before any widgets with resolution-independent data are created. See the **XmNunitType** resource description in the reference pages for **XmGadget**, **XmManager**, and **XmPrimitive** for more information on resolution independence.

This function sets the font units for all screens on the display.

NOTE: XmSetFontUnits is obsolete and exists for compatibility with previous releases. Instead of using this function, provide initial values or call **XtSetValues** for the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

display	Defines the display for which this font unit value is to be applied.
h_value	Specifies the value to be used for horizontal units in the conversion calculations.
h_value	Specifies the value to be used for vertical units in the conversion calculations.

Related Information

XmConvertUnits(3), **XmSetFontUnit**(3), **XmGadget**(3), **XmManager**(3), **XmPrimitive**(3), and **XmScreen**(3).

XmSetMenuCursor

Purpose

A function that modifies the menu cursor for a client

Synopsis

#include <Xm/Xm.h>

void XmSetMenuCursor(
        Display * *display***,**
        Cursor *cursorId***);**

Description

XmSetMenuCursor programmatically modifies the menu cursor for a client; after the cursor has been created by the client, this function registers the cursor with the menu system. After calling this function, the specified cursor is displayed whenever this client displays a Motif menu on the indicated display. The client can then specify different cursors on different displays.

This function sets the menu cursor for all screens on the display. **XmSetMenuCursor** is obsolete and exists for compatibility with previous releases. Instead of using this function, provide initial values or call **XtSetValues** for the XmScreen resource **XmNmenuCursor**.

display	Specifies the display to which the cursor is to be associated
cursorId	Specifies the **X** cursor ID

Related Information

XmScreen(3).

XmSetProtocolHooks

Purpose

A VendorShell function that allows preactions and postactions to be executed when a protocol message is received from MWM

Synopsis

#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmSetProtocolHooks(
    Widget *shell***,**
    Atom *property***,**
    Atom *protocol***,**
    XtCallbackProc *prehook***,**
    XtPointer *pre_closure***,**
    XtCallbackProc *posthook***,**
    XtPointer *post_closure***);**

Description

XmSetProtocolHooks is used by shells that want to have preactions and postactions executed when a protocol message is received from MWM. Since there is no guaranteed ordering in execution of event handlers or callback lists, this allows the shell to control the flow while leaving the protocol manager structures opaque.

XmSetWMProtocolHooks is a convenience interface. It calls **XmSetProtocolHooks** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
property	Specifies the protocol property
protocol	Specifies the protocol atom
prehook	Specifies the procedure to call before calling entries on the client callback list
pre_closure	Specifies the client data to be passed to the prehook when it is invoked

posthook	Specifies the procedure to call after calling entries on the client callback list
post_closure	Specifies the client data to be passed to the posthook when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmInternAtom**(3), and **XmSetWMProtocolHooks**(3).

XmSetWMProtocolHooks

Purpose

A VendorShell convenience interface that allows preactions and postactions to be executed when a protocol message is received from the window manager

Synopsis

#include <Xm/Xm.h>
#include <Xm/Protocols.h>

void XmSetWMProtocolHooks(
  Widget *shell*,
  Atom *protocol*,
  XtCallbackProc *prehook*,
  XtPointer *pre_closure*,
  XtCallbackProc *posthook*,
  XtPointer *post_closure*);

Description

XmSetWMProtocolHooks is a convenience interface. It calls **XmSetProtocolHooks** with the property value set to the atom returned by interning WM_PROTOCOLS.

shell	Specifies the widget with which the protocol property is associated
protocol	Specifies the protocol atom (or an *int* cast to **Atom**)
prehook	Specifies the procedure to call before calling entries on the client callback list
pre_closure	Specifies the client data to be passed to the prehook when it is invoked
posthook	Specifies the procedure to call after calling entries on the client callback list
post_closure	Specifies the client data to be passed to the posthook when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell**(3).

Related Information

VendorShell(3), **XmInternAtom**(3), and **XmSetProtocolHooks**(3).

XmSpinBox

Purpose

The SpinBox widget class

Synopsis

#include <Xm/SpinB.h>

Description

SpinBox allows the user to select a value from a ring of related but mutually exclusive choices which are displayed in sequence. The SpinBox always has an increment arrow, a decrement arrow, and one or more other children. The choices are displayed, one at a time, in a traversable text child (**XmText** or **XmTextField**. The user clicks Btn1 on an arrow to display the next (or previous) item in the ring of choices. By pressing and holding Btn1 on an arrow, the user continuously cycles through the choices.

The traversable children in a SpinBox can be of type **XmNUMERIC** or **XmSTRING**, as defined by the **XmNspinBoxChildType** constraint resource. The ring of choices for numeric children is defined by minimum, maximum, incremental, and decimal point values. The ring of choices for string children is defined in an array of compound strings.

The application programmer can include multiple traversable children in the SpinBox. For example, a SpinBox might consist of a pair of arrows and month, day, and year text fields. The arrows only spin the child that currently has focus.

Arrow size is specified by the SpinBox resource **XmNarrowSize**. This value sets both width and height of each arrow in pixels.

The programmer can display SpinBox arrows in one of several layouts, as specified by the **XmNarrowLayout** resource:

XmARROWS_BEGINNING
Places a pair of left and right arrows before the children.

XmARROWS_END
Places a pair of left and right arrows after the children.

XmARROWS_SPLIT
Places one arrow on each side of the children.

XmARROWS_FLAT_BEGINNING
Places a pair of arrows side by side before the *XmSpinBox* children.

XmARROWS_FLAT_BEGINNING
Places a pair of arrows side by side after the *XmSpinBox* children.

Positions for **XmARROWS_BEGINNING** and **XmARROWS_END** are dependent on the **VendorShell** resource **XmNlayoutDirection**. When layout direction is left-to-right, beginning arrows are positioned to the left of the children. When layout direction is right-to-left, beginning arrows are positioned to the right.

The actions of the arrows are determined by the **VendorShell** resource **XmNlayoutDirection**. For left-to-right layouts, the right arrow is the increment arrow and the left arrow is the decrement arrow. For right-to-left layouts, the right arrow is the decrement arrow and the left arrow is the increment arrow.

For a numeric type child, the increment arrow increases the displayed value by the incremental value up to the maximum. The decrement arrow decreases the displayed value by the given incremental value down to the minimum.

The increment arrow for a string type child moves toward the last entry of the array of compound strings (by increasing the SpinBox constraint resource **XmNposition**). The decrement arrow moves toward the first entry of the compound string array.

The programmer can also control the sensitivity of each arrow in the SpinBox. Sensitive arrows spin choices; insensitive arrows do not spin choices. Arrow sensitivity is set for the SpinBox widget by using the **XmNdefaultArrowSensitivity** resource, but it can be modified on a per child basis by using the **XmNarrowSensitivity** constraint resource.

SpinBox provides two callbacks to application programmers. (In addition, the callbacks of the SpinBox's children may be invoked.) Each of these callbacks receives a pointer to **XmSpinBoxCallbackStruct**. The **XmNmodifyVerifyCallback** procedures are called *before* a new choice is displayed. The **XmNvalueChangedCallback** procedures are called *after* a new choice is displayed.

XmNmodifyVerifyCallback tells the application what the new position will be in the ring of choices. This callback can be used to make the SpinBox stop at the upper and lower limits or go to a different, nonconsecutive choice. The application allows the change in position by leaving the *doit* member set to True. The application can spin to a position other than the next consecutive position by leaving *doit* set to True and by changing the *position* member to the desired position. When *doit* is set to False by an application, there is no change in the choice displayed.

After a new choice is displayed, the **XmNvalueChangedCallback** procedure is called. The application can use this procedure to perform tasks when specific values are reached or when boundaries are crossed. For example, if the user spins from January back to December, the application could change to the previous year. If the user spins from December to January, the application could change to the next year.

SpinBox dimensions can be set using the Core resources **XmNheight** and **XmNwidth**. If dimensions are not specified, the SpinBox size is determined by the sizes of its arrows and children. The SpinBox will attempt to grow so that the arrows and all children are visible.

SpinBox uses the *XmQTaccessTextual* trait and holds the *XmQTnavigator* trait.

Classes

SpinBox inherits behavior, resources, and traits from the **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is *xmSpinBoxWidgetClass*.

The class name is **XmSpinBox**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate whether the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmSpinBox Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNarrowLayout	XmCArrowLayout	unsigned char	XmARROWS_-BEGINNING	CSG
XmNarrowOrientation	XmCArrowOrientation	unsigned char	XmARROWS_-VERTICAL	CSG

XmNarrowSize	XmCArrowSize	Dimension	16	CSG
XmNdefaultArrow-Sensitivity	XmCDefaultArrow-Sensitivity	unsigned char	XmARROWS_-SENSITIVE	CSG
XmNdetailShadow-Thickness	XmCDetailShadow-Thickness	Dimension	2	CSG
XmNinitialDelay	XmCInitialDelay	unsigned int	250 ms	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	dynamic	CSG
XmNmodifyVerify-Callback	XmCCallback	XtCallbackList	NULL	C
XmNrepeatDelay	XmCRepeatDelay	unsigned int	200 ms	CSG
XmNspacing	XmCSpacing	Dimension	dynamic	CSG
XmNvalueChanged-Callback	XmCCallback	XtCallbackList	NULL	C

XmNarrowLayout

Specifies placement of the two arrows in the widget. Possible layouts are as follows:

XmARROWS_BEGINNING

Places left and right arrows beside each other, before the child(ren). Positioning for this layout is dependent on the VendorShell resource **XmNlayoutDirection.**

XmARROWS_END

Places left and right arrows beside each other, after the child(ren). Positioning for this layout is dependent on the VendorShell resource **XmNlayoutDirection**.

XmARROWS_FLAT_BEGINNING

Places a pair of arrows side by side before the *XmSpinBox* children. Positioning for this layout is dependent on the VendorShell resource **XmNlayoutDirection.**

XmARROWS_FLAT_END

Places a pair of arrows side by side after the *XmSpinBox* children. Positioning for this layout is dependent on the VendorShell resource **XmNlayoutDirection.**

XmARROWS_SPLIT
Places a left arrow on the left side and a right arrow on the right side of the child(ren).

XmNarrowSize
Specifies both the width and height of the arrow in pixels.

XmNdefaultArrowSensitivity
Specifies the default sensitivity of the arrows in the widget. Insensitive arrows change color, cannot be depressed, and perform no action. (This resource may be overridden by the constraint resource **XmNarrowSensitivity** for individual traversable text children of the SpinBox.) Possible default sensitivity values are as follows:

XmARROWS_SENSITIVE
Both arrows are sensitive.

XmARROWS_DECREMENT_SENSITIVE
Only the decrement arrow (as determined by **XmNlayoutDirection**) is sensitive. The increment arrow is insensitive.

XmARROWS_INCREMENT_SENSITIVE
Only the increment arrow (as determined by **XmNlayoutDirection**) is sensitive. The decrement arrow is insensitive.

XmARROWS_INSENSITIVE
Both arrows are insensitive.

XmNdetailShadowThickness
Specifies the thickness of the inside arrow shadows. The default thickness is 2 pixels.

XmNinitialDelay
Specifies how long, in milliseconds, the mouse button must be held down before automatic spinning begins. In other words, when the user selects the increment or decrement arrow and keeps it depressed, this delay occurs before the choices start spinning. If **XmNinitialDelay** is 0, then **XmNrepeatDelay** is used as the initial delay.

XmNmarginHeight
Specifies the amount of blank space between the top edge of the SpinBox widget and the first item in each column, and the bottom edge of the SpinBox widget and the last item in each column.

XmNmarginWidth
Specifies the amount of blank space between the left edge of the SpinBox widget and the first item in each row, and the right edge of the SpinBox widget and the last item in each row.

XmNmodifyVerifyCallback
This callback is called before the SpinBox position changes (see the Constraint resource **XmNposition**). The application can use this callback to set the next position, change SpinBox resources, or cancel the impending action. For example, this callback can be used to stop the spinning just before wrapping at the upper and lower position boundaries. If the *doit* member is set to False, nothing happens. Otherwise the position changes. Reasons sent by the callback are **XmCR_SPIN_NEXT**, **XmCR_SPIN_PRIOR**, **XmCR_SPIN_FIRST**, or **XmCR_SPIN_LAST**.

XmNrepeatDelay
When the user selects and keeps an arrow button depressed by pressing and holding Btn1, spinning begins. After the time specified in **XmNinitialDelay** elapses, the SpinBox position changes automatically until the arrow button is released. The **XmNrepeatDelay** resource specifies the delay in milliseconds between each automatic change. If **XmNrepeatDelay** is set to 0 (zero), automatic spinning is turned off and **XmNinitialDelay** is ignored.

XmNspacing
Specifies the horizontal and vertical spacing between items contained within the SpinBox widget.

XmNvalueChangedCallback
This is called *n*+1 times for *n* SpinBox position changes (see the Constraint resource **XmNposition**). Reasons sent by the callback are **XmCR_OK**, **XmCR_SPIN_NEXT**, **XmCR_SPIN_PRIOR**, **XmCR_SPIN_FIRST**, or **XmCR_SPIN_LAST**. Other members are detailed in the callback structure description.

XmSpinBox Constraint Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNarrowSensitivity	XmCArrowSensitivity	unsigned char	XmARROWS_DEFAULT_- SENSITIVITY	CSG
XmNdecimalPoints	XmCDecimalPoints	short	0	CSG
XmNincrementValue	XmCIncrementValue	int	1	CSG
XmNmaximumValue	XmCMaximumValue	int	10	CSG
XmNminimumValue	XmCMinimumValue	int	0	CSG
XmNnumValues	XmCNumValues	int	0	CSG
XmNposition	XmCPosition	int	0	CSG
XmNpositionType	XmCPositionType	char	XmPOSITION_- VALUE	CG
XmNspinBoxChildType	XmSpinBoxChildType	unsigned char	XmSTRING	CG
XmNvalues	XmCValues	XmStringTable	NULL	CSG

XmNarrowSensitivity

Specifies the sensitivity of the arrows for a SpinBox child. By using this resource in the definition of a SpinBox child, the application programmer can override the default SpinBox sensitivity (set by **XmNdefaultArrowSensitivity**) for a particular child. This allows each traversable child to have a different arrow sensitivity. The arrow sensitivity values are as follows:

XmARROWS_SENSITIVE

Both arrows are sensitive.

XmARROWS_DECREMENT_SENSITIVE

Only the decrement arrow (as determined by **XmNlayoutDirection**) is sensitive.

XmARROWS_INCREMENT_SENSITIVE

Only the increment arrow (as determined by **XmNlayoutDirection**) is sensitive.

XmARROWS_INSENSITIVE

Both arrows are insensitive.

XmARROWS_DEFAULT_SENSITIVITY

Use the sensitivity specified in the **XmNdefaultArrowSensitivity** resource.

XmNdecimalPoints

Specifies the number of decimal places used when displaying the value of a SpinBox numeric type child. If the number of decimal places specified is greater than the number of digits in a displayed value, the value is padded with 0 (zeros). For example, when *XmNinitialValue* is 1 and **XmNmaximumValue** is 1000 and **XmNdecimalPoints** is 3, the range of values displayed in the SpinBox is 0.001 to 1.000. This is used only when **XmNspinBoxChildType** is **XmNUMERIC**.

XmNincrementValue

Specifies the amount by which to increment or decrement a SpinBox numeric type child. This is used only when **XmNspinBoxChildType** is **XmNUMERIC**.

XmNmaximumValue

Specifies the highest possible value for a numeric SpinBox. This is used only when **XmNspinBoxChildType** is **XmNUMERIC**.

XmNminimumValue

Specifies the lowest possible value for a numeric SpinBox. This is used only when **XmNspinBoxChildType** is **XmNUMERIC**.

XmNnumValues

Specifies the number of strings in **XmNvalues**. The application must change this value when strings are added or removed from **XmNvalues**. This is used only when **XmNspinBoxChildType** is **XmSTRING**.

XmNposition

Specifies the position of the currently displayed item. The interpritation of *XmNposition* is dependent upon the value of the *XmNpositionType* resource.

When *XmNpositionType* is *XmPOSITION_INDEX* the *XmNposition* value is interpreted as follows: For *XmSpinBox* children of type *XmNUMERIC*, the *XmNposition* resource is interpreted as an index into an array of items. The minimum allowable value for *XmNposition* is 0. The maximum allowable value for *XmNposition* is **(XmNmaximumValue-XmNminimumValue)/ XmNincrementValue**. The value display by the *XmSpinBox* child is **XmNminimumValue+(XmNposition*XmNincrementValue)**. For *XmSpinBox* children of type *XmSTRING*, the *XmNposition* resource is interpreted as an index into an array of *XmNnumValues* items. The minimum allowable value for *XmNposition* is 0. The maximum allowable value for *XmNposition* is **XmNnumValues - 1**. The value displayed by the *XmSpinBox* is the *XmNposition*'th value in the *XmNvalues* array.

When *XmNpositionType* is *XmPOSITION_VALUE* the *XmNposition* value is interpreted as follows:

For *XmSpinBox* children of type *XmNUMERIC*, the *XmNposition* resource is interpreted as the actual value to be displayed. The minimum allowable value for *XmNposition* is *XmNminimumValue*. The maximum allowable value for *XmNposition* is *XmNmaximumValue*. The value displayed by the *XmSpinBox* child is *XmNposition*. For *XmSpinBox* children of type *XmSTRING*, the interpretation is the same for *XmPOSITION_VALUE* as for *XmPOSITION_INDEX*.

Position values falling outside the specified range are invalid. When an application assigns a value to *XmNposition* which is less than the minimum, *XmNposition* is set to the minimum and an error message is displayed. When an application assigns a value to *XmNposition* which is greater than the maximum, *XmNposition* is set to the maximum and an error message is displayed.

XmNpositionType
Specifies how values the *XmNposition* resource are to be interpreted. Valid values include *XmPOSITION_INDEX* and *XmPOSITION_VALUE*.

XmNspinBoxChildType
Specifies the type of data displayed in the child:

XmNUMERIC
The SpinBox choice range is defined by numeric minimum, maximum, and incremental values.

XmSTRING
The SpinBox choices are alphanumeric.

XmNvalues Specifies the array of **XmString**s to be displayed in a SpinBox string type child. The application must change **XmNnumValues** when strings are added to or removed from **XmNvalues**. This is used only when **XmNspinBoxChildType** is **XmSTRING**.

Inherited Resources

SpinBox inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G
XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback

A pointer to the following structure is passed to each callback:

```
typedef struct
{
        int reason;
        XEvent * event;
        Widget widget;
        Boolean doit;
        int position;
        XmString value;
        Boolean crossed_boundary;
} XmSpinBoxCallbackStruct;
```

reason Indicates why the callback was invoked. Reasons may be the following:

XmCR_OK Spinning has stopped because the SpinBox arrow has been disarmed. **XmCR_OK** is either the last or only call.

XmCR_SPIN_NEXT
The increment arrow has been armed and position is increasing. Further callbacks will come. For a numeric type child, the values displayed are approaching the maximum. For a string SpinBox, the values displayed are approaching the last entry in the array of **XmString** s.

XmCR_SPIN_PRIOR
The decrement arrow has been armed and position is decreasing. Further callbacks will come. For a numeric type child, the values displayed are approaching the minimum. For a string type child, the values displayed are approaching the first entry in the array of **XmString**s.

XmCR_SPIN_FIRST
The begin data (osfBeginData) key sequence has been pressed. The SpinBox is at its first position, displaying the lowest value or the first entry in the array of **XmString**s.

XmCR_SPIN_LAST
The end data (osfEndData) key sequence has been pressed. The SpinBox is at its last position, displaying the highest value or the last entry in the array of **XmString**s.

event Points to the *XEvent* that triggered this callback.

widget Specifies the child widget affected by this callback.

doit When the callback is **XmNmodifyVerifyCallback**, *doit* indicates whether or not an action will be performed before the SpinBox position changes. If the callback leaves *doit* set to True (the default), the spinning action is performed. If the callback sets *doit* to

False, the spinning action is not performed. When the callback is **XmNvalueChangedCallback**, *doit* is ignored.

position Specifies the next value of the SpinBox position (same as **XmNposition**). This is an output field for the **XmNmodifyVerifyCallback**, which may change the next position as dictated by the needs of an application.

value Specifies the new **XmString** value in the text child widget. The user program must copy this string if it is to be used outside the callback routine.

crossed_boundary
Specifies whether or not the SpinBox has crossed the upper or lower boundary (the last or first compound string, or the maximum or minimum value). The *crossed_boundary* value is True if the SpinBox has just crossed a boundary, and False if it has not.

Translations

The **XmSpinBox** translations are as follows:

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<Btn1Down>:
SpinBArm()

<Btn1Up>:
SpinBDisarm()

:<Key>osfUp :
SpinBPrior()

:<Key>osfDown :
SpinBNext()

:<Key>osfLeft :
SpinBLeft()

:<Key>osfRight :
SpinBRight()

:<Key>osfBeginData :
SpinBFirst()

:<Key>osfEndData :
SpinBLast()

Accelerators

The **XmNaccelerators** resource of a SpinBox are added to each traversable text child. The default **XmNaccelerators** are defined in the following list. The bindings for **<Key>osfUp** and **<Key>osfDown** cannot be changed.

<Key> osfUp:
SpinBPrior()

<Key> osfDown:
SpinBNext()

KeyUp osfUp:
SpinBDisarm()

KeyUp osfDown:
SpinBDisarm()

<Key> osfLeft:
SpinBLeft()

<Key> osfRight:
SpinBRight()

KeyUp osfLeft:
SpinBDisarm()

KeyUp osfRight:
SpinBDisarm()

<Key> osfBeginData:
SpinBFirst()

<Key> osfEndData:
SpinBLast()

Action Routines

The **XmSpinBox** action routines are as follows:

SpinBArm(): Visually arms the SpinBox by drawing the armed arrow so that it appears to be depressed. This action is initiated when the user presses Btn1 while the pointer is within the boundaries of either the increment or decrement arrow. The arrow remains visually armed as long as Btn1 remains depressed.

If the time period specified by **XmNrepeatDelay** is not greater than zero milliseconds, nothing else happens while Btn1 remains depressed.

If the time period specified by **XmNrepeatDelay** is greater than zero milliseconds, and the arrow is disarmed before the time period specified by **XmNinitialDelay** has elapsed, nothing else happens in this action.

If the time period specified by **XmNrepeatDelay** is greater than zero milliseconds, and the arrow is still armed after the time period specified by **XmNinitialDelay** has elapsed, the following occurs:

- The *reason* member of the SpinBox callback structure, **XmSpinBoxCallbackStruct**, is set to **XmCR_SPIN_NEXT** if the increment arrow is armed, or to **XmCR_SPIN_PRIOR** if the decrement arrow is armed.
- The *position* member is set to the next position.
- The *doit* member is set to True.
- **XmNmodifyVerifyCallback**, if it exists, is invoked. The application may change the value of *position* and *doit*. If the application sets *doit* to False, nothing else happens until the **XmNrepeatDelay** period has elapsed, or until Btn1 is released.

If *doit* remains set to True, the following occurs:

- The value of **XmNposition** is changed to the value of *position* in the SpinBox callback structure.
- The text corresponding to the new position is displayed in the traversable text child that currently has focus.
- The *reason* member of the SpinBox callback structure is set to **XmCR_SPIN_NEXT** if the increment arrow is armed, or **XmCR_SPIN_PRIOR** if the decrement arrow is armed.
- The *position* member is set to the current (new) value of **XmNposition**.

- **XmNvalueChangedCallback**, if it exists, is called. SpinBox ignores any changes to *position* or *doit* members made by **XmNvalueChangedCallback**.

These events are repeated each time the **XmNrepeatDelay** period elapses and the arrow remains armed.

SpinBDisarm():
Visually disarms the SpinBox by drawing the previously armed arrow so that it no longer appears to be depressed.

If the time period specified by **XmNrepeatDelay** is not greater than zero milliseconds, or the time period specified by **XmNinitialDelay** has not elapsed, the following then occurs:

- The *reason* member of the SpinBox callback structure, **XmSpinBoxCallbackStruct**, is set to **XmCR_SPIN_NEXT** if the increment arrow is armed, or to **XmCR_SPIN_PRIOR** if the decrement arrow is armed.
- The *position* member is set to the next position.
- The *doit* member is set to True.
- The **XmNmodifyVerifyCallback**, if there is one, is invoked. The application may change the value of *position* and *doit*. If the application sets *doit* to False, nothing else happens until the **XmNrepeatDelay** period has elapsed, or until Btn1 is released.

If *doit* remains set to True, the following occurs:

- The value of **XmNposition** is changed to the value of *position* in the SpinBox callback structure.
- The text corresponding to the new position is displayed in the traversable text child that currently has focus.
- The *reason* member of the SpinBox callback structure is set to **XmCR_SPIN_NEXT** if the increment arrow is armed, or **XmCR_SPIN_PRIOR** if the decrement arrow is armed.
- The *position* member is set to the current (new) value of **XmNposition**.

- **XmNvalueChangedCallback**, if it exists, is called. SpinBox ignores any changes to *position* or *doit* members made by an **XmNvalueChangedCallback**.

If an **XmNvalueChangedCallback** procedure is issued after the button has been armed, regardless of the value of **XmNrepeatDelay** or whether the **XmNinitialDelay** has expired:

- The *reason* member of the SpinBox callback structure is set to **XmCR_OK**.
- The *position* member is set to the current value of **XmNposition**.
- **XmNvalueChangedCallback**, if it exists, is called.

SpinBFirst():

The following occurs:

- The *reason* member of the SpinBox callback structure, **XmSpinBoxCallbackStruct**, is set to **XmCR_SPIN_FIRST**.
- The *position* member is set to the first (0) position.
- The *doit* member is set to True.
- **XmNmodifyVerifyCallback**, if it exists, is invoked. The application may change the value of *position* and *doit*. If the application sets *doit* to False, nothing else happens until the **XmNrepeatDelay** period has elapsed, or until Btn1 is released.

If *doit* remains set to True, the following occurs:

- The value of **XmNposition** is changed to the value of *position* in the SpinBox callback structure.
- The text corresponding to the new position is displayed in the traversable text child that currently has focus.
- The *reason* member of the SpinBox callback structure is set to **XmCR_SPIN_FIRST**.
- The *position* member is set to the current (new) value of **XmNposition**.
- **XmNvalueChangedCallback**, if it exists, is called.
- The *reason* member of the SpinBox callback structure is set to **XmCR_OK**.

- The *position* member is set to the current (new) **XmNposition** value.
- The **XmNvalueChangedCallback** is called again. SpinBox ignores any changes to *position* or *doit* members made by **XmNvalueChangedCallback**.

SpinBLast(): The following occurs:

- The *reason* member of the SpinBox callback structure, **XmSpinBoxCallbackStruct**, is set to **XmCR_SPIN_LAST**.
- The *position* member is set to the last position.
- The *doit* member is set to True.
- **XmNmodifyVerifyCallback**, if it exists, is invoked. The application may change the value of *position* and *doit*. If the application sets *doit* to False, nothing else happens until the **XmNrepeatDelay** period has elapsed, or until Btn1 is released.

If *doit* remains set to True, the following occurs:

- The value of **XmNposition** is changed to the value of *position* in the SpinBox callback structure.
- The text corresponding to the new position is displayed in the traversable text child that currently has focus.
- The *reason* member of the SpinBox callback structure is set to **XmCR_SPIN_LAST**.
- The *position* member is set to the current (new) value **XmNposition**.
- **XmNvalueChangedCallback**, if it exists, is called.
- The *reason* member of the SpinBox callback structure is set to **XmCR_OK**.
- The *position* member is set to the current (new) of **XmNposition**.
- **XmNvalueChangedCallback** is called again. SpinBox ignores any changes to the *position* or *doit* members made by **XmNvalueChangedCallback**.

SpinBLeft(): If the VendorShell resource **XmNlayoutDirection** is left-to-right, the **SpinBPrior** action is invoked. Otherwise, the **SpinBNext** action is invoked.

SpinBNext():

Visually arms the SpinBox by drawing the increment arrow so that it appears to be depressed. The following occurs:

- The *reason* member of the SpinBox callback structure, **XmSpinBoxCallbackStruct**, is set to **XmCR_SPIN_NEXT**.
- The *position* member is set to the next position.
- The *doit* member is set to True.
- **XmNmodifyVerifyCallback**, if it exists, is invoked. The application may change the value of *position* and *doit*. If the application sets *doit* to False, nothing else happens until the **XmNrepeatDelay** period has elapsed, or until Btn1 is released.

If *doit* remains set to True, the following occurs:

- The value of **XmNposition** is changed to the value of *position* in the SpinBox callback structure.
- The text corresponding to the new position is displayed in the traversable text child that currently has focus.
- The *reason* member of the SpinBox callback structure is set to **XmCR_SPIN_NEXT**.
- The *position* member is set to the current (new) value of **XmNposition**.
- **XmNvalueChangedCallback**, if it exists, is called.
- The *reason* member of the SpinBox callback structure is set to **XmCR_OK**.
- The *position* member is set to the current (new) **XmNposition**.
- The **XmNvalueChangedCallback** is called again. SpinBox ignores any changes to *position* or *doit* members made by **XmNvalueChangedCallback**.

SpinBPrior():

Visually arms the SpinBox by drawing the decrement arrow so that it appears to be depressed. The following occurs:

- The *reason* member of the SpinBox callback structure, **XmSpinBoxCallbackStruct**, is set to **XmCR_SPIN_PRIOR**.

- The *position* member is set to the next position.
- The *doit* member is set to True.
- **XmNmodifyVerifyCallback**, if it exists, is invoked. The application may change the value of *position* and *doit*. If the application sets *doit* to False, nothing else happens until the **XmNrepeatDelay** period has elapsed, or until Btn1 is released.

If *doit* remains set to True, the following occurs:

- The value of **XmNposition** is changed to the value of *position* in the SpinBox callback structure.
- The text corresponding to the new position is displayed in the traversable text child that currently has focus.
- The *reason* member of the SpinBox callback structure is set to **XmCR_SPIN_PRIOR**.
- The *position* member is set to the current (new) value of **XmNposition**.
- **XmNvalueChangedCallback**, if it exists, is called.
- The *reason* member of the SpinBox callback structure is set to **XmCR_OK**.
- The *position* member is set to the current (new) value of **XmNposition**.
- **XmNvalueChangedCallback** is called again. SpinBox ignores any changes to *position* or *doit* members made by **XmNvalueChangedCallback**.

SpinBRight():
If the VendorShell resource **XmNlayoutDirection** is left-to-right, the **SpinBNext** action is invoked. Otherwise, the **SpinBPrior** action is invoked.

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateSpinBox**(3), **XmManager**(3), and **XmString**(3).

XmSimpleSpinBoxAddItem

Purpose

add an item to the XmSimpleSpinBox

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxAddItem(
    Widget *w***,**
    XmString *item***,**
    int *pos***);**

Description

The **XmSimpleSpinBoxAddItem** function adds the given item to the XmSimpleSpinBox at the given position.

The *w* argument specifies the widget ID.

The *item* argument specifies the **XmString** for the new item.

The *pos* argument specifies the position of the new item.

Return Values

The **XmSimpleSpinBoxAddItem** function returns no value.

Related Information

XmSimpleSpinBox(3),

XmSimpleSpinBoxDeletePos(3), **XmSimpleSpinBoxSetItem**(3).

XmSimpleSpinBoxDeletePos

Purpose

delete a XmSimpleSpinBox item

Synopsis

```
#include <Xm/SpinB.h>

void XmSimpleSpinBoxDeletePos(
        Widget w,
        int pos);
```

Description

The **XmSimpleSpinBoxDeletePos** function deletes a specified item from a XmSimpleSpinBox widget.

The *w* argument specifies the widget ID.

The *pos* argument specifies the position of the item to be deleted. A value of 1 means the first item in the list; zero means the last item.

Return Values

The **XmSimpleSpinBoxDeletePos** function returns no value.

Related Information

XmSimpleSpinBox(3),

XmSimpleSpinBoxAddItem(3), **XmSimpleSpinBoxSetItem**(3).

XmSimpleSpinBoxSetItem

Purpose

set an item in the XmSimpleSpinBox list

Synopsis

#include <Xm/SpinB.h>

void XmSimpleSpinBoxSetItem(
    Widget *w*,
    XmString *item***);**

Description

The **XmSimpleSpinBoxSetItem** function selects an item in the list of the given XmSimpleSpinBox widget and makes it the current value.

The *w* argument specifies the widget ID.

The *item* argument specifies the **XmString** for the item to be set in the XmSimpleSpinBox. If the *item* is not found on the list, **XmSimpleSpinBoxSetItem** notifies the user via the **XtWarning** function.

Return Values

The **XmSimpleSpinBoxSetItem** function returns no value.

Related Information

XmSimpleSpinBox(3),

XmSimpleSpinBoxAddItem(3), **XmSimpleSpinBoxDeletePos**(3); **XtWarning**(3).
in the CAE Specification, Window Management: X Toolkit Intrinsics.

XmSpinBoxValidatePosition

Purpose

translate the current value of the specified XmSpinBox child into a valid position

Synopsis

#include <Xm/SpinBox.h>

int XmSpinBoxValidatePosition(
  Widget *textfield***,**
  int **position***);**

Description

The **XmSpinBoxValidatePosition** function is a utility that can be used by applications wanting to implement a policy for tracking user modifications to editable **XmSpinBox** children of type *XmNUMERIC*. The specifics of when and how the user's modifications take effect is left up to the application.

text_field The *text_field* argument specifies the widget ID of the child of the **XmSpinBox** that is being modified. The requirement on *text_field* is that it holds the **accessTextual** trait (already a requirement for children of **XmSpinBox**). This way, **XmSpinBox** can extract the string out of the *text_field* widget (even if it is not an *XmTextField*).

position The location pointed to by the position argument is assigned the result of the translation done by **XmSpinBoxValidatePosition**. **XmSpinBoxValidatePosition** first checks to make sure this is an *XmNUMERIC* **XmSpinBox** child. If it is not, **XSmpinBoxValidatePosition** sets position to the current position and returns *XmCURRENT_VALUE*.

XmSpinBoxValidatePosition attempts to translate the input string to a floating point number. If this translation fails, **XmSpinBoxValidatePosition** sets position to the current position and returns *XmCURRENT_VALUE*.

XmSpinBoxValidatePosition converts the floating point number to an integer using the *XmNdecimalPoints* resource. Extra decimal places are truncated. The resulting integer is range checked to make sure it falls within the valid range defined by

XmNminimumValue and *XmNmaximumValue* inclusive. If the input falls outside this range, **XmSpinBoxValidatePosition** sets position to the nearest limit and returns either *XmMINIMUM_VALUE* or *XmMAXIMUM_VALUE*.

Finally, **XmSpinBoxValidatePosition** checks the integer to make sure it belongs to the series defined by **XmNminimumValue ... XmNminumumValue + ((n - 1) * XmNincrementlValue)**. If the integer does not belong to this series, **XmSpinBoxValidatePosition** sets position to the nearest element which is less than or equal to the integer and returns *XmINCREMENT_VALUE*.

Otherwise, **XmSpinBoxValidatePosition** assigns the integer to position and returns *XmVALID_VALUE*.

Return Values

The **XmSpinBoxValidatePosition** function returns the status of the validation. The set of possible values returned is as follows:

XmCURRENT_VALUE
Cannot convert, returning current position_value.

XmMINIMUM_VALUE
Less than min.

XmMAXIMUM_VALUE
More than max.

XmINCREMENT_VALUE
Not on increment.

XmVALID_VALUE
Okay.

Examples

This first example demonstrates how the **XmSpinBoxValidatePosition** function could be used from inside an **XmNmodifyVerifyCallback** callback installed on the **XmSpinBox** or the **XmSimpleSpinBox**:

```
/*
 * Install a callback on a spin box arrow press.
 */
```

```
  XtAddCallback(sb, XmNmodifyVerifyCallback, ModifyVerifyCB, NULL);
  XtAddCallback(simple_sb, XmNmodifyVerifyCallback, ModifyVerifyCB, NULL);
```

with the callback doing:

```
void ModifyVerifyCB(widget, call_data, client_data) {
    XmSpinBoxCallbackStruct *cbs = (XmSpinBoxCallbackStruct*) call_data;
    int position;
    Widget textual = NULL;
    if (XtIsSubclass(w, xmSimpleSpinBoxWidgetClass))
    {
        Arg args[1];
        XtSetArg(args[0], XmNtextField, &textual);
        XtGetValues(w, args, 1);
    }
    else if (XtIsSubclass(w, xmSpinBoxWidgetClass))
      textual = cbs->widget;
    else
      textual = (Widget) NULL;

    ...

    if (XmSpinBoxValidatePosition(textual, &position) == XmCURRENT_VALUE)
      XBell(XtDisplay(w), 0);
    else
      cbs->position = position;
}
```

This second example demonstrates how the **XmSpinBoxValidatePosition** function could be used from inside an **XmNactivateCallback** callback installed on the **TextField** child of the **XmSpinBox**:

```
/*
 * Install a callback on a spin box arrow press.
 */
XtAddCallback(tf, XmNactivateCallback, ModifyVerifyChildCB, NULL);
```

with the callback doing:

```
void ModifyVerifyChildCB(widget, call_data, client_data) {
    int     position;
    Widget  textual = widget;
    Arg     args[1];

    if (XmSpinBoxValidatePosition (textual, &position) == XmCURRENT_VALUE)
      XBell(XtDisplay(widget), 0);

    /* Set the position constraint resource of the textfield */

    XtSetArg(args[0], XmNposition, position);
    XtSetValues(textual, args, 1);
}
```

Related Information

XmSpinBox(3), **XmCreateSpinBox**(3)

XmSimpleSpinBox

Purpose

a simple SpinBox widget class

Synopsis

#include <Xm/SSpinB.h>

Description

The XmSimpleSpinBox widget is a user interface control to increment and decrement an arbitrary TextField. For example, it can be used to cycle through the months of the year or days of the month.

Widget subclassing is not supported for the XmSimpleSpinBox widget class.

Classes

The XmSimpleSpinBox widget inherits behavior and resources from the **Core**, **Composite** and **XmManager** classes.

The class pointer is *XmSimpleSpinBoxWidgetClass*.

The class name is **XmSimpleSpinBoxWidget**.

New Resources

The following table defines a set of widget resources used by the application to specify data. The application can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, the application must remove the *XmN* or *XmC* prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, the application must remove the **Xm** prefix and use the remaining letters (in either lower case or upper case, but including any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmSimpleSpinBox Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNarrowLayout	**XmCArrowLayout**	**unsigned char**	XmARROWS_-END	CSG
XmNarrowSensitivity	**XmCArrow- Sensitivity**	**unsigned char**	XmARROWS_-SENSITIVE	CSG
XmNcolumns	**XmCColumn**	**short**	20	CSG
XmNdecimalPoints	**XmCDecimalPoints**	**short**	0	CSG
XmNeditable	**XmCEditable**	**Boolean**	True	CSG
XmNincrementValue	**XmCIncrementValue**	**int**	1	CSG
XmNinitialDelay	**XmCInitialDelay**	**unsigned int**	250	CSG
XmNmaximumValue	**XmCMaximumValue**	**int**	10	CSG
XmNminimumValue	**XmCMinimumValue**	**int**	0	CSG
XmNmodifyVerify-Callback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNnumValues	**XmCNumValues**	**int**	0	CSG
XmNposition	**XmCPosition**	**int**	0	CSG
XmNrepeatDelay	**XmCRepeatDelay**	**unsigned int**	200	CSG
XmNspinBoxChildType	**XmCSpinBox-ChildType**	**unsigned char**	XmSTRING	CG
XmNtextField	**XmCTextField**	**Widget**	dynamic	G
XmNvalueChanged-Callback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNvalues	**XmCValues**	**XmStringTable**	NULL	CSG

XmNarrowLayout

Specifies the style and position of the SpinBox arrows. The following values are supported:

XmARROWS_FLAT_BEGINNING
: The arrows are placed side by side to the right of the TextField.

XmARROWS_FLAT_END
: The arrows are placed side by side to the left of the TextField.

XmARROWS_SPLIT
The down arrow is on the left and the up arrow is on the right of the TextField.

XmARROWS_BEGINNING
The arrows are stacked and placed on the left of the TextField.

XmARROWS_END
The arrows are stacked and placed on the right of the TextField.

XmNarrowSensitivity
Specifies the sensitivity of the arrows in the XmSimpleSpinBox. The following values are supported:

XmARROWS_SENSITIVE
Both arrows are active to user selection.

XmARROWS_DECREMENT_SENSITIVE
The down arrow is active and the up arrow is inactive to user selection.

XmARROWS_INCREMENT_SENSITIVE
The up arrow is active and the down arrow is inactive to user selection.

XmARROWS_INSENSITIVE
Both arrows are inactive to user selection.

XmNcolumns
Specifies the number of columns of the text field.

XmNdecimalPoints
Specifies the position of the radix character within the numeric value when **XmNspinBoxChildType** is **XmNUMERIC**. This resource is used to allow for floating point values in the XmSimpleSpinBox widget.

XmNeditable
Specifies whether the text field can take input.

When **XmNeditable** is used on a widget it sets the dropsite to **XmDROP_SITE_ACTIVE**.

XmNincrementValue
Specifies the amount to increment or decrement the **XmNposition** when the **XmNspinBoxChildType** is **XmNUMERIC**. When the Up action is activated, the **XmNincrementValue** is added to the **XmNposition** value; when the Down action is activated, the **XmNincrementValue** is subtracted from the **XmNposition** value. When **XmNspinBoxChildType** is **XmSTRING**, this resource is ignored.

XmNinitialDelay
Specifies the amount of time in milliseconds before the Arrow buttons will begin to spin continuously.

XmNnumValues
Specifies the number of items in the **XmNvalues** list when the **XmNspinBoxChildType** resource is **XmSTRING**. The value of this resource must be a positive integer. The **XmNnumValues** is maintained by the XmSimpleSpinBox widget when items are added or deleted from the **XmNvalues** list. When **XmNspinBoxChildType** is not **XmSTRING**, this resource is ignored.

XmNvalues Supplies the list of strings to cycle through when the *XmNspinButtonChildType* resource is **XmSTRING**. When **XmNspinBoxChildType** is not **XmSTRING**, this resource is ignored.

XmNmaximumValue
Specifies the upper bound on the XmSimpleSpinBox's range when **XmNspinBoxChildType** is **XmNUMERIC**.

XmNminimumValue
Specifies the lower bound on the XmSimpleSpinBox's range when **XmNspinBoxChildType** is **XmNUMERIC**.

XmNmodifyVerifyCallback
Specifies the callback to be invoked just before the XmSimpleSpinBox position changes. The application can use this callback to implement new application-related logic (including setting new position spinning to, or canceling the impending action). For example, this callback can be used to stop the spinning just before wrapping at the upper and lower position boundaries. If the application sets the *doit* member of the **XmSimpleSpinBoxCallbackStruct** to False, nothing happens.

Otherwise, the position changes. Reasons sent by the callback are **XmCR_SPIN_NEXT**, or **XmCR_SPIN_PRIOR**.

XmNposition

The **XmNposition** resource has a different value based on the **XmNspinBoxChildType** resource. When **XmNspinBoxChildType** is **XmSTRING**, the **XmNposition** is the index into the **XmNvalues** list for the current item. When the **XmNspinBoxChildType** resource is **XmNUMERIC**, the **XmNposition** is the integer value of the XmSimpleSpinBox that falls within the range of **XmNmaximumValue** and **XmNminimumValue**.

XmNrepeatDelay

Specifies the number of milliseconds between repeated calls to the **XmNvalueChangedCallback** while the user is spinning the XmSimpleSpinBox.

XmNspinBoxChildType

Specifies the style of the XmSimpleSpinBox. The following values are supported:

XmSTRING

The child is a string value that is specified through the **XmNvalues** resource and incremented and decremented by changing the **XmNposition** resource.

XmNUMERIC

The child is a numeric value that is specified through the **XmNposition** resource and incremented according to the **XmNincrementValue** resource.

XmtextField

Specifies the textfield widget.

XmNvalueChangedCallback

Specifies the callback to be invoked whenever the value of the **XmNposition** resource is changed through the use of the spinner arrows. The **XmNvalueChangedCallback** passes the **XmSimpleSpinBoxCallbackStruct** *call_data* structure.

Inherited Resources

The XmSimpleSpinBox widget inherits behavior and resources from the following named superclasses. For a complete description of each resource, see the man page for that superclass.

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	**XmCBottomShadow-Color**	**Pixel**	dynamic	CSG
XmNbottomShadow-Pixmap	**XmCBottomShadow-Pixmap**	**Pixmap**	**XmUNSPECIFIED_-PIXMAP**	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightPixmap	**XmCHighlight-Pixmap**	**Pixmap**	dynamic	CSG
XmNinitialFocus	**XmCInitialFocus**	**Widget**	NULL	CSG
XmNnavigationType	**XmCNavigation- Type**	**XmNavigation-Type**	dynamic	CSG
XmNshadowThickness	**XmCShadow-Thickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmString-Direction**	dynamic	CG
XmNtopShadowColor	**XmCTopShadow-Color**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadow-Pixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	CSG
XmNunitType	**XmCUnitType**	**unsigned char**	dynamic	CSG
XmNuserData	**XmCUserData**	**XtPointer**	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	**XmCReadOnly**	**WidgetList**	NULL	G
XmNinsertPosition	**XmCInsertPosition**	**XtOrderProc**	default procedure	CSG
XmNnumChildren	**XmCReadOnly**	**Cardinal**	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	**XmCAccelerators**	**XtAccelerators**	dynamic	CSG
XmNancestorSensitive	**XmCSensitive**	**Boolean**	dynamic	G
XmNbackground	**XmCBackground**	**Pixel**	dynamic	CSG
XmNbackgroundPixmap	**XmCPixmap**	**Pixmap**	**XmUNSPECIFIED_-PIXMAP**	CSG
XmNborderColor	**XmCBorderColor**	**Pixel**	XtDefaultForeground	CSG
XmNborderPixmap	**XmCPixmap**	**Pixmap**	**XmUNSPECIFIED_-PIXMAP**	CSG
XmNborderWidth	**XmCBorderWidth**	**Dimension**	0	CSG
XmNcolormap	**XmCColormap**	**Colormap**	dynamic	CG
XmNdepth	**XmCDepth**	**int**	dynamic	CG
XmNdestroyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNheight	**XmCHeight**	**Dimension**	dynamic	CSG
XmNinitialResources-Persistent	**XmCInitialResources-Persistent**	**Boolean**	True	C
XmNmapped-WhenManaged	**XmCMappedWhen-Managed**	**Boolean**	True	CSG
XmNscreen	**XmCScreen**	**Screen ***	dynamic	CG
XmNsensitive	**XmCSensitive**	**Boolean**	True	CSG
XmNtranslations	**XmCTranslations**	**XtTranslations**	dynamic	CSG
XmNwidth	**XmCWidth**	**Dimension**	dynamic	CSG
XmNx	**XmCPosition**	**Position**	0	CSG
XmNy	**XmCPosition**	**Position**	0	CSG

Callback Information

A pointer to the following structure is passed to each XmSimpleSpinBox callback:

```
typedef struct {
        int             reason;
        XEvent          *event;
        Widget          widget;
        Boolean doit;
        int             position;
        XmString        value;
        Boolean crossed_boundary;
} XmSimpleSpinBoxCallbackStruct;
```

The *reason* argument indicates why the callback was invoked. There are three possible reasons for this callback to be issued. The reason is **XmCR_OK** when this is the first call to the callback at the beginning of a spin or if it is a single activation of the spin arrows. If the XmSimpleSpinBox is in the process of being continuously spun, then the reason will be **XmCR_SPIN_NEXT** or **XmCR_SPIN_PRIOR**, depending on the arrow that is spinning.

The *event* argument points to the **XEvent** that triggered the callback. It can be **NULL** when the XmSimpleSpinBox is continuously spinning.

The *widget* argument is the widget identifier for the simple spin box widget that has been affected by this callback.

The *doit* argument is set only when the *call_data* comes from the **XmNmodifyVerifyCallback**. It indicates that the action that caused the callback to be called should be performed. The action is not performed if *doit* is set to False.

The *position* argument is the new value of the **XmNposition** resource as a result of the spin.

The *value* argument is the new **XmString** value displayed in the Text widget as a result of the spin. The application must copy this string if it is used beyond the scope of the *call_data* structure.

The *crossed_boundary* argument is True when the spinbox cycles. This is the case when a **XmNspinBoxChildType** of **XmSTRING** wraps from the first item to the last or the last item to the first. In the case of the **XmNspinBoxChildType** of **XmNUMERIC**, the boundary is crossed when the XmSimpleSpinBox cycles from the maximum value to the minimum or vice versa.

Errors/Warnings

The toolkit will display a warning if the application tries to set the value of the **XmNtextField** resource, which is read-only (marked G in the resource table).

Related Information

XmSpinBox(3), **XmCreateSpinBox**(3), **XmSimpleSpinBoxAddItem**(3), **XmSimpleSpinBoxDeletePos**(3), **XmSimpleSpinBoxSetItem**(3), **Composite**(3), **Core**(3), **XmManager**(3), **XmText**(3), **XmTextField**(3), **XtGetValues**(3), **XtSetValues**(3)

XmStringBaseline

Purpose

A compound string function that returns the number of pixels between the top of the character box and the baseline of the first line of text

Synopsis

#include <Xm/Xm.h>

Dimension XmStringBaseline(
    XmRenderTable *rendertable*,
    XmString *string*);

Description

XmStringBaseline returns the number of pixels between the top of the character box and the baseline of the first line of text in the provided compound string.

rendertable	Specifies the render table
string	Specifies the string

Return Values

Returns the number of pixels between the top of the character box and the baseline of the first line of text.

Related Information

XmStringCreate(3).

XmStringByteCompare

Purpose

A compound string function that indicates the results of a byte-by-byte comparison

Synopsis

#include <Xm/Xm.h>

Boolean XmStringByteCompare(
        XmString *s1***,**
        XmString *s2***);**

Description

This function is obsolete and exists for compatibility with previous releases. **XmStringByteCompare** returns a Boolean indicating the results of a byte-by-byte comparison of two compound strings.

In general, if two compound strings are created with the same (**char ***) string using **XmStringCreateLocalized** in the same language environment, the compound strings compare as equal. If two compound strings are created with the same (**char ***) string and the same font list element tag set other than **XmFONTLIST_DEFAULT_TAG** using **XmStringCreate**, the strings compare as equal.

In some cases, once a compound string is put into a widget, that string is converted into an internal form to allow faster processing. Part of the conversion process strips out unnecessary or redundant information. If an application then does an **XtGetValues** to retrieve a compound string from a widget (specifically, Label and all of its subclasses), it is not guaranteed that the compound string returned is byte-for-byte the same as the string given to the widget originally.

s1	Specifies a compound string to be compared with *s2*
s2	Specifies a compound string to be compared with *s1*

Return Values

Returns True if two compound strings are identical byte-by-byte.

Related Information

XmStringCreate(3) and **XmStringCreateLocalized**(3).

XmStringByteStreamLength

Purpose

A function that returns the size of a string

Synopsis

#include <Xm/Xm.h>
unsigned int XmStringByteStreamLength (*string***)**
        unsigned char   ****string*;**

Description

XmStringByteStreamLength receives a byte stream format string and returns the size, in bytes, of that stream, including the header. Because of this header information, even a NULL *string* will cause **XmStringByteStreamLength** to return a non-zero value.

string	Specifies the byte stream format string.

Return Values

Returns the size of *string*, including the header.

XmStringCompare

Purpose

A compound string function that compares two strings

Synopsis

#include <Xm/Xm.h>

Boolean XmStringCompare(
    XmString *s1*,
    XmString *s2*);

Description

XmStringCompare returns a Boolean value indicating the results of a semantically equivalent comparison of two compound strings.

Semantically equivalent means that the strings have the same text components, font list element tags, directions, and separators. In general, if two compound strings are created with the same (**char ***) string using **XmStringCreateLocalized** in the same language environment, the compound strings compare as equal. If two compound strings are created with the same text and tag argument using **XmStringCreate**, the strings compare as equal.

s1	Specifies a compound string to be compared with *s2*
s2	Specifies a compound string to be compared with *s1*

Return Values

Returns True if two compound strings are equivalent.

Related Information

XmStringCreate(3) and **XmStringCreateLocalized**(3).

XmStringComponentCreate

Purpose

A compound string function that creates arbitrary components

Synopsis

#include <Xm/Xm.h>

XmString XmStringComponentCreate(
XmStringComponentType *c_type***,**
unsigned int *length***,**
XtPointer *value***);**

Description

XmStringComponentCreate creates a new **XmString** component of type *c_type*, containing *value*. If *value* is invalid for the particular component type, this function fails and returns NULL.

c_type	Specifies the type of component to be created.
length	Specifies the length in bytes of *value*. Note that this must be precisely the length of the *value* string, *not* including any trailing null characters.
value	Specifies the value to be used in the creation of the component.

Refer to the **XmStringComponentType**(3) reference page for a list of the possible **XmString** component types.

Return Values

If *value* is invalid for *c_type*, fails and returns NULL. Otherwise, this function returns a new compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

Related Information

XmString(3), **XmStringGetNextTriple**, **XmStringComponentType**, and **XmStringFree**(3).

XmStringComponentType

Purpose

Data type for compound string components

Synopsis

#include <Xm/Xm.h>

Description

XmStringComponentType is the data type specifying compound string component types. A compound string component identifies some part of a compound string, and can have a value and length. A compound string component can be one of the following types. These component types are grouped according to their length and value types.

The following components have values of NULL and lengths of 0 (zero).

XmSTRING_COMPONENT_SEPARATOR
This component usually maps to a newline or carriage return in displayed text.

XmSTRING_COMPONENT_TAB
This component may be thought of as a text component containing only a single tab.

XmSTRING_COMPONENT_LAYOUT_POP
The layout direction is kept on a stack, with the current direction kept on top of the stack. When this component is read, the most recently read layout direction is popped off the stack and replaced by the direction immediately before it.

XmSTRING_COMPONENT_END
This component marks the end of a compound string. No other components should follow. If an application does not place an **XmSTRING_COMPONENT_END** component at the end of an **XmString**, Motif automatically does it for the application.

The following component has a value of **XmDirection** and the length of that direction.

XmSTRING_COMPONENT_LAYOUT_PUSH
: The layout direction is kept on a stack, with the current direction kept on top of the stack. This component replaces the current layout direction, and causes another to be pushed onto the top of this stack.

The following component has a value of **XmStringDirection** and the length of that direction.

XmSTRING_COMPONENT_DIRECTION
: This component sets the string direction by overriding the previous string direction.

The following components have values of type **char *** or some equivalent type, and the lengths of these types.

XmSTRING_COMPONENT_LOCALE_TEXT
: This component contains the multibyte text of a compound string.

XmSTRING_COMPONENT_WIDECHAR_TEXT
: This component contains the widechar text of a compound string.

XmSTRING_COMPONENT_TEXT
: This component contains the charset text of a compound string. Note that a compound string cannot contain both charset and locale (multibyte or widechar) text.

XmSTRING_COMPONENT_RENDITION_BEGIN
: This component marks the beginning of a new rendition. All text following this component will be rendered using this rendition as the primary one. If there is already a rendition in effect, it is kept in memory and used to fill in any unspecified values in the primary rendition. Renditions are kept until a corresponding **XmSTRING_COMPONENT_RENDITION_END** component is encountered.

XmSTRING_COMPONENT_RENDITION_END
: This component signifies that the specified rendition will no longer be used to render text, and will not be available to fill in unspecified values of newer renditions.

XmSTRING_COMPONENT_UNKNOWN
: This component type signifies that the component contents belong to an unknown component type.

XmSTRING_COMPONENT_LOCALE
Use this component to specify the locale in which an internationalized application is to execute. The only valid character string for this component is **_MOTIF_DEFAULT_LOCALE**.

XmSTRING_COMPONENT_TAG
For charset text, this is the tag of the font to be used to display the text. This tag is sometimes referred to as the charset tag or the fontlist tag.

XmSTRING_COMPONENT_CHARSET
This component is obsolete and remains for compatibility with previous releases. It has been replaced by **XmSTRING_COMPONENT_TAG**.

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
This component is obsolete and remains for compatibility with previous releases. It has been replaced by **XmSTRING_COMPONENT_TAG**.

Some compound string components depend on values defined in other components. The **XmSTRING_COMPONENT_TAB** component definition, for example, depends on information in the **XmSTRING_COMPONENT_RENDITION_BEGIN**. To account for these dependencies, a typical compound string will have its member components in the following order:

```
[
  [ XmSTRING_COMPONENT_LAYOUT_PUSH ]
  [ XmSTRING_COMPONENT_RENDITION_BEGIN ]*
  [ XmSTRING_COMPONENT_TAG | XmSTRING_COMPONENT_LOCALE ]
  [ XmSTRING_COMPONENT_TAB ]*
  [ XmSTRING_COMPONENT_DIRECTION ]
  [ XmSTRING_COMPONENT_TEXT |
    XmSTRING_COMPONENT_LOCALE_TEXT |
    XmSTRING_COMPONENT_WIDECHAR_TEXT ]
  [ XmSTRING_COMPONENT_RENDITION_END ]*
  [ XmSTRING_COMPONENT_LAYOUT_POP ]
  [ XmSTRING_COMPONENT_SEPARATOR ]
]*
XmSTRING_COMPONENT_END
```

XmStringConcat

Purpose

A compound string function that appends one string to another

Synopsis

```
#include <Xm/Xm.h>

XmString XmStringConcat(
        XmString s1,
        XmString s2);
```

Description

XmStringConcat copies *s2* to the end of *s1* and returns a copy of the resulting compound string. The original strings are preserved. The function will allocate space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

s1	Specifies the compound string to which a copy of *s2* is appended
s2	Specifies the compound string that is appended to the end of *s1*

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3) and **XmStringFree**(3).

XmStringConcatAndFree

Purpose

A compound string function that appends one string to another and frees the original strings

Synopsis

#include <Xm/Xm.h>

XmString XmStringConcatAndFree(
        XmString *s1*,
        XmString *s2*);

Description

XmStringConcatAndFree copies *s2* to the end of *s1* and returns a copy of the resulting compound string. The original strings are freed. The function will allocate space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

s1	Specifies the compound string to which a copy of *s2* is appended
s2	Specifies the compound string that is appended to the end of *s1*

The **XmStringConcatAndFree** function works like the **XmStringConcat** function, except that it frees the *s1* and *s2* strings, and is therefore more efficient. You should use **XmStringConcatAndFree** instead of **XmStringConcat** if you want *s1* and *s2* to be freed afterwards.

Return Values

Returns a new compound string.

Related Information

XmStringConcat(3), **XmStringCreate**(3), and **XmStringFree**(3).

XmStringCopy

Purpose

A compound string function that makes a copy of a string

Synopsis

#include <Xm/Xm.h>

XmString XmStringCopy(
        XmString *s1***);**

Description

XmStringCopy makes a copy of an existing compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

s1	Specifies the compound string to be copied

Return Values

Returns a compound string.

Related Information

XmStringCreate(3) and **XmStringFree**(3).

XmStringCreate

Purpose

A compound string function that creates a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringCreate(
        char ****text*,**
        char ****tag*);**

Description

XmStringCreate creates a compound string with two components: text and a font list element tag. The function will allocate space to hold the returned compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

text	Specifies a NULL-terminated string to be used as the text component of the compound string.
tag	Specifies the tag component to be associated with the given text. The value **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

Return Values

Returns a new compound string.

Related Information

XmFontList(3), **XmFontListAdd**(3), **XmFontListAppendEntry**(3), **XmFontListCopy**(3), **XmFontListCreate**(3), **XmFontListEntryCreate**(3), **XmFontListEntryFree**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryGetTag**(3), **XmFontListEntryLoad**(3), **XmFontListFree**(3),

XmFontListFreeFontContext(3), **XmFontListGetNextFont**(3), **XmFontListInitFontContext**(3), **XmFontListNextEntry**(3), **XmFontListRemoveEntry**(3), **XmString**(3), **XmStringBaseline**(3), **XmStringByteCompare**(3), **XmStringCompare**(3), **XmStringConcat**(3), **XmStringCopy**(3), **XmStringCreateLocalized**(3), **XmStringCreateLtoR**(3), **XmStringCreateSimple**(3), **XmStringDirection**(3), **XmStringDirectionCreate**(3), **XmStringDraw**(3), **XmStringDrawImage**(3), **XmStringDrawUnderline**(3), **XmStringEmpty**(3), **XmStringExtent**(3), **XmStringFree**(3), **XmStringFreeContext**(3), **XmStringGetLtoR**(3), **XmStringGetNextComponent**(3), **XmStringGetNextSegment**(3), **XmStringHasSubstring**(3), **XmStringHeight**(3), **XmStringInitContext**(3), **XmStringLength**(3), **XmStringLineCount**(3), **XmStringNConcat**(3), **XmStringNCopy**(3), **XmStringPeekNextComponent**(3), **XmStringSegmentCreate**(3), **XmStringSeparatorCreate**(3), **XmStringTable**(3), and **XmStringWidth**(3).

XmStringCreateLocalized

Purpose

A compound string function that creates a compound string in the current locale

Synopsis

#include <Xm/Xm.h>

XmString XmStringCreateLocalized(
        char ****text*);**

Description

XmStringCreateLocalized creates a compound string containing the specified text in the current language environment. An identical compound string would result from the function **XmStringCreate** called with **XmFONTLIST_DEFAULT_TAG** explicitly as the tag component.

The function will allocate space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

text Specifies a NULL-terminated string of text encoded in the current language environment to be used as the text component of the compound string

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3).

XmStringCreateLtoR

Purpose

A compound string function that creates a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringCreateLtoR(
    char ***_text_,**
    char ***_tag_);**

Description

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmStringGenerate**. **XmStringCreateLtoR** creates a compound string with two components: text and a tag component. This function scans for **\n** characters in the text. When one is found, the text up to that point is put into a segment followed by a separator component. No final separator component is appended to the end of the compound string. The direction component defaults to left-to-right. This function assumes that the encoding is single byte rather than multibyte.

The function will allocate space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

text Specifies a NULL-terminated string to be used as the text component of the compound string.

tag Specifies the tag component to be associated with the given text. The value **XmFONTLIST_DEFAULT_TAG** is retained for compatibility with previous releases.

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3) and **XmStringGenerate**(3).

XmStringCreateSimple

Purpose

A compound string function that creates a compound string in the language environment of a widget

Synopsis

#include <Xm/Xm.h>

XmString XmStringCreateSimple(
        char * *text*);

Description

XmStringCreateSimple creates a compound string with a text component and a charset tag. It derives the character set from the current language environment.

The routine attempts to derive a character set from the value of the LANG environment variable. If this does not result in a valid character set, the routine uses a vendor-specific default. If the vendor has not specified a different value, this default is ISO8859-1.

The function will allocate space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

NOTE: This routine is obsolete and exists for compatibility with previous releases. It has been replaced by **XmStringCreateLocalized**.

text Specifies a NULL-terminated string to be used as the text component of the compound string.

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3) and **XmStringCreateLocalized**(3).

XmStringDirectionCreate

Purpose

A compound string function that creates a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringDirectionCreate(
    XmStringDirection *direction***);**

Description

XmStringDirectionCreate creates a compound string with a single component, a direction with the given value. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

direction Specifies the value of the direction component. The possible values are:

XmSTRING_DIRECTION_L_TO_R
Specifies left to right display.

XmSTRING_DIRECTION_R_TO_L
Specifies right to left display.

XmSTRING_DIRECTION_DEFAULT
Specifies that the display direction will be set by the widget in which the compound string is to be displayed.

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3).

XmStringDirectionToDirection

Purpose

A function that converts from XmStringDirection to XmDirection

Synopsis

#include <Xm/Xm.h>

XmDirection XmStringDirectionToDirection(
        XmStringDirection *direction***);**

Description

XmStringDirectionToDirection converts the specified **XmStringDirection** direction value to its equivalent **XmDirection** value. This function provides backward compatibility with the **XmStringDirection** data type.

direction   Specifies the **XmStringDirection** value to be converted.

Return Values

Returns the following **XmDirection** values:

XmLEFT_TO_RIGHT
    If the *direction* argument is **XmSTRING_DIRECTION_L_TO_R**.

XmRIGHT_TO_LEFT
    If the *direction* argument is **XmSTRING_DIRECTION_R_TO_L**.

XmDEFAULT_DIRECTION
    If the *direction* argument was not either of the above.

Related Information

XmStringDirection(3) and **XmDirection**(3).

XmStringDraw

Purpose

A compound string function that draws a compound string in an X window

Synopsis

#include <Xm/Xm.h>

void XmStringDraw(
  Display * *d***,**
  Window *w***,**
  XmRenderTable *rendertable***,**
  XmString *string***,**
  GC *gc***,**
  Position *x***,**
  Position *y***,**
  Dimension *width***,**
  unsigned char *alignment***,**
  unsigned char *layout_direction***,**
  XRectangle * *clip***);**

Description

XmStringDraw draws a compound string in an X Window. If a compound string segment uses a rendition that contains a font set, the graphic context passed to this routine will have the GC font member left in an undefined state. The underlying **XmbStringDraw** function called by this routine modifies the font ID field of the GC passed into it and does not attempt to restore the font ID to the incoming value. If the compound string segment is not drawn using a font set, the graphic context must contain a valid font member. Graphic contexts created by **XtGetGC** are not valid for this routine; instead, use **XtAllocateGC** to create a graphic context.

d	Specifies the display.
w	Specifies the window.
rendertable	Specifies the render table.
string	Specifies the string.

gc Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that will contain the displayed compound string.

y Specifies a coordinate of the rectangle that will contain the displayed compound string.

width Specifies the width of the rectangle that will contain the displayed compound string.

alignment Specifies how the string will be aligned within the specified rectangle. It is either **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, or **XmALIGNMENT_END**.

layout_direction
Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string will be drawn. If the value is NULL, clipping will be determined by the GC.

Related Information

XmStringCreate(3).

XmStringDrawImage

Purpose

A compound string function that draws a compound string in an X Window and creates an image

Synopsis

#include <Xm/Xm.h>

void XmStringDrawImage(
    Display * *d***,**
    Window *w***,**
    XmRenderTable *rendertable***,**
    XmString *string***,**
    GC *gc***,**
    Position *x***,**
    Position *y***,**
    Dimension *width***,**
    unsigned char *alignment***,**
    unsigned char *layout_direction***,**
    XRectangle * *clip***);**

Description

XmStringDrawImage draws a compound string in an X Window and paints both the foreground and background bits of each character. If a compound string segment uses a rendition that contains a font set, the graphic context passed to this routine will have the GC font member left in an undefined state. The underlying **XmbStringDraw** function called by this routine modifies the font ID field of the GC passed into it and does not attempt to restore the font ID to the incoming value. If the compound string segment is not drawn using a font set, the graphic context must contain a valid font member. Graphic contexts created by **XtGetGC** are not accepted by this routine; instead, use **XtAllocateGC** to create a graphic context.

d	Specifies the display.
w	Specifies the window.

rendertable Specifies the render table.

string Specifies the string.

gc Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that will contain the displayed compound string.

y Specifies a coordinate of the rectangle that will contain the displayed compound string.

width Specifies the width of the rectangle that will contain the displayed compound string.

alignment Specifies how the string will be aligned within the specified rectangle. It is either **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, or **XmALIGNMENT_END**.

layout_direction
Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string will be drawn. If NULL, clipping will be determined by the GC.

Related Information

XmStringCreate(3).

XmStringDrawUnderline

Purpose

A compound string function that underlines a string drawn in an X Window

Synopsis

#include <Xm/Xm.h>

void XmStringDrawUnderline(
 Display * *d*,
 Window *w*,
 XmRenderTable *rendertable*,
 XmString *string*,
 GC *gc*,
 Position *x*,
 Position *y*,
 Dimension *width*,
 unsigned char *alignment*,
 unsigned char *layout_direction*,
 XRectangle * *clip*,
 XmString *underline***);**

Description

XmStringDrawUnderline draws a compound string in an X Window. If the substring identified by *underline* can be matched in *string*, the substring will be underlined. Once a match has occurred, no further matches or underlining will be done. Only the first text component of *underline* is used for matching.

If a compound string segment uses a rendition that contains a font set, the graphic context passed to this routine will have the GC font member left in an undefined state. The underlying **XmbStringDraw** function called by this routine modifies the font ID field of the GC passed into it and does not attempt to restore the font ID to the incoming value. If the compound string segment is not drawn using a font set, the graphic context must contain a valid font member. Graphic contexts created by **XtGetGC** are not accepted by this routine; instead, use **XtAllocateGC** to create a graphic context.

d	Specifies the display.
w	Specifies the window.
rendertable	Specifies the render table.
string	Specifies the string.
gc	Specifies the graphics context to use.
x	Specifies a coordinate of the rectangle that will contain the displayed compound string.
y	Specifies a coordinate of the rectangle that will contain the displayed compound string.
width	Specifies the width of the rectangle that will contain the displayed compound string.
alignment	Specifies how the string will be aligned within the specified rectangle. It is one of **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, or **XmALIGNMENT_END**.
layout_direction	Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.
clip	Allows the application to restrict the area into which the compound string will be drawn. If it is NULL, clipping will be determined by the GC.
underline	Specifies the substring to be underlined.

Related Information

XmStringCreate(3).

XmStringEmpty

Purpose

A compound string function that provides information on the existence of non-zero-length text components

Synopsis

#include <Xm/Xm.h>

Boolean XmStringEmpty(
  XmString *s1***);**

Description

XmStringEmpty returns a Boolean value indicating whether any non-zero-length text components exist in the provided compound string. It returns True if there are no text segments in the string. If this routine is passed NULL as the string, it returns True.

s1	Specifies the compound string

Return Values

Returns True if there are no text segments in the string. If this routine is passed NULL as the string, it returns True.

Related Information

XmStringCreate(3).

XmStringExtent

Purpose

A compound string function that determines the size of the smallest rectangle that will enclose the compound string

Synopsis

#include <Xm/Xm.h>

void XmStringExtent(
    XmRenderTable *rendertable***,**
    XmString *string***,**
    Dimension **width***,**
    Dimension **height***);**

Description

XmStringExtent determines the width and height, in pixels, of the smallest rectangle that will enclose the provided compound string.

rendertable	Specifies the render table
string	Specifies the string
width	Specifies a pointer to the width of the rectangle
height	Specifies a pointer to the height of the rectangle

Related Information

XmStringCreate(3).

XmStringFree

Purpose

A compound string function that conditionally deallocates memory

Synopsis

#include <Xm/Xm.h>

void XmStringFree(
    XmString *string***);**

Description

XmStringFree conditionally recovers memory used by a compound string. Applications should call **XmStringFree** when the application no longer needs *string*.

string	Specifies the compound string to be freed

Related Information

XmStringCreate(3).

XmStringFreeContext

Purpose

A compound string function that releases the string scanning context data structure

Synopsis

```
#include <Xm/Xm.h>

void XmStringFreeContext(
        XmStringContext context);
```

Description

XmStringFreeContext releases the string scanning context data structure.

context	Specifies the string context structure that was allocated by the **XmStringInitContext** function

Related Information

XmStringCreate(3) and **XmStringInitContext**(3).

XmStringGenerate

Purpose

A convenience function that generates a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringGenerate(
    XtPointer *text*,
    XmStringTag *tag*,
    XmTextType *type*,
    XmStringTag *rendition*);

Description

XmStringGenerate calls the **XmStringParseText** function with a default parse table of entries consisting of ’\n’, which maps to Separator, and ’\t’, which maps to Tab. Matching *RENDITION_BEGIN* and *RENDITION_END* components containing *rendition* are placed around the resulting **XmString**.

text Specifies a NULL-terminated string containing characters of a type determined by *type*.

tag Specifies the tag to be used in creating the result. The type of tag created (charset or locale) depends on the text type and the value given. If specified value is NULL, and *type* indicates that a charset tag should be created, then the tag will have the value of **XmFONTLIST_DEFAULT_TAG**. If *tag* is NULL, and *type* indicates a locale tag, then the tag will have the value of **_MOTIF_DEFAULT_LOCALE**.

type Specifies the type of text to be passed in, and the tag type. If a locale tag should be created, then *type* has a value of either **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**. If a charset should be created, *type* has a value of **XmCHARSET_TEXT**.

rendition Specifies the rendition tag to be used in an **XmSTRING_COMPONENT_RENDITION_BEGIN** component which will begin the returned string and in an **XmSTRING_COMPONENT_RENDITION_END** component which will end it. If *rendition* is NULL, no rendition tag is placed.

Return Values

Returns a new compound string. The function will allocate space to hold the returned compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

Related Information

XmString(3) and **XmStringFree**(3).

XmStringGetLtoR

Purpose

A compound string function that searches for a text segment in the input compound string

Synopsis

#include <Xm/Xm.h>

Boolean XmStringGetLtoR(
XmString *string***,**
XmStringCharSet *tag***,**
char *****text*);**

Description

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmStringUnparse**. **XmStringGetLtoR** returns the first text component in the input compound string that is tagged with the given tag component. The returned text is to be a NULL-terminated sequence of single byte characters. If the function returns True, the function will allocate space to hold the returned *text*. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XtFree**.

string	Specifies the compound string.
tag	Specifies the font list element tag associated with the text. A value of **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.
text	Specifies a pointer to a NULL terminated string.

Return Values

Returns True if the matching text segment can be found. On return, *text* will have a NULL terminated byte sequence containing the matched segment.

Related Information

XmStringCreate(3).

XmStringGetNextComponent

Purpose

A compound string function that returns the type and value of the next component in a compound string

Synopsis

#include <Xm/Xm.h>

XmStringComponentType XmStringGetNextComponent(
  XmStringContext *context***,**
  char *****text*,**
  XmStringTag ****tag*,**
  XmStringDirection ****direction*,**
  XmStringComponentType ****unknown_tag*,**
  unsigned short ****unknown_length*,**
  unsigned char *****unknown_value*);**

Description

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmStringGetNextTriple**. **XmStringGetNextComponent** returns the type and value of the next component in the compound string identified by *context*. Components are returned one at a time. On return, only some output parameters will be valid; which ones can be determined by examining the returned component type. The following table describes the valid returns for each component type.

Valid Fields	**Component Type**
tag	*XmSTRING_COMPONENT_LOCALE*, *XmSTRING_COMPONENT_TAG*
text	*XmSTRING_COMPONENT_LOCALE_TEXT*, *XmSTRING_COMPONENT_TEXT*, *XmSTRING_COMPONENT_WIDECHAR_TEXT*
direction	*XmSTRING_COMPONENT_DIRECTION*

unknown_tag, *unknown_length*, *unknown_value*	*XmSTRING_COMPONENT_LAYOUT_POP*, *XmSTRING_COMPONENT_LAYOUT_PUSH*, *XmSTRING_COMPONENT_TAB*, *XmSTRING_COMPONENT_RENDITION_BEGIN*, *XmSTRING_COMPONENT_RENDITION_END*
no valid field	*XmSTRING_COMPONENT_SEPARATOR*, *XmSTRING_COMPONENT_END*, *XmSTRING_COMPONENT_UNKNOWN*

Note that several components produce a return value of **XmSTRING_COMPONENT_UNKNOWN**. The data returned by these components is returned in the *unknown_tag*, *unknown_length*, and *unknown_value* fields. This apparent inconsistency is designed to accomodate older applications that may not be equipped to handle the newer component types of Motif version 2.0 and beyond. Consequently, the use of this procedure is not recommended. Instead, use the **XmStringGetNextTriple** procedure, which provides all the functionality of **XmStringGetNextComponent**, and is fully compatible with the newer component types.

If the function return value is **XmSTRING_COMPONENT_LOCALE_TEXT** or **XmSTRING_COMPONENT_TEXT**, the the function allocates space to hold the returned *text*. If the function return value is **XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG**, or **XmSTRING_COMPONENT_TAG**, then the function allocates space to hold the returned *tag*. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XtFree**.

context Specifies the string context structure that was allocated by the **XmStringInitContext** function.

text Specifies a pointer to a NULL terminated string.

tag Specifies a pointer to the tag component associated with the text. The value **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

direction Specifies a pointer to the direction of the text.

unknown_tag
Specifies a pointer to the tag of an unknown component.

unknown_length
Specifies a pointer to the length of an unknown component.

unknown_value
Specifies a pointer to the value of an unknown component.

Return Values

Returns the type of component found. Refer to the **XmStringComponentType**(3) reference page for a list of component types.

Related Information

XmStringComponentType(3), **XmStringCreate**(3), and **XmStringInitContext**(3).

XmStringGetNextSegment

Purpose

A compound string function that fetches the bytes in the next segment of a compound string

Synopsis

#include <Xm/Xm.h>

Boolean XmStringGetNextSegment(
  XmStringContext *context***,**
  char *****text*,**
  XmStringTag ****tag*,**
  XmStringDirection ****direction*,**
  Boolean ****separator*);**

Description

This routine is obsolete and exists for compatibility with previous releases. To read the contents of a compound string, read each component of the string with **XmStringGetNextTriple**. This **XmString** function returns the type, length, and value of the next component in the compound string. **XmStringGetNextSegment** fetches the bytes in the next segment; repeated calls fetch sequential segments. The *text*, *tag*, and *direction* of the fetched segment are returned each time. A Boolean status is returned to indicate whether a valid segment was successfully parsed.

If the function returns True, then the function allocates space to hold the returned *text* and *tag*. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XtFree**.

context	Specifies the string context structure which was allocated by the **XmStringInitContext** function
text	Specifies a pointer to a NULL-terminated string
tag	Specifies a pointer to the font list element tag associated with the text
direction	Specifies a pointer to the direction of the text

separator	Specifies whether the next component of the compound string is a separator

Return Values

Returns True if a valid segment is found.

Related Information

XmStringCreate(3) and **XmStringInitContext**(3).

XmStringGetNextTriple

Purpose

An XmString function that returns the type, length, and value of the next component in the compound string

Synopsis

#include <Xm/Xm.h>

XmStringComponentType XmStringGetNextTriple(
    XmStringContext *context***,**
    unsigned int **length***,**
    XtPointer **value***);**

Description

XmStringGetNextTriple returns the type, length, and value of the next component in the compound string identified by *context*. This function returns one component at a time.

context Specifies the string context structure that was allocated by the **XmStringInitContext** function.

length Specifies a pointer to the length of the value of the returned component.

value Specifies a pointer to the value of the returned component. If the returned value is not NULL, the function allocates space to hold the returned value. When the application no longer needs the returned compound string, the application should call **XtFree**.

Return Values

Returns the type of the component found. Refer to the **XmStringComponentType**(3) reference page for a list of component types.

Related Information

XmDirection(3), **XmString**(3), **XmStringComponentType**(3), **XmStringGetNextComponent**(3), and **XmStringPeekNextTriple**(3).

XmStringHasSubstring

Purpose

A compound string function that indicates whether one compound string is contained within another

Synopsis

```
#include <Xm/Xm.h>

Boolean XmStringHasSubstring(
        XmString string,
        XmString substring);
```

Description

XmStringHasSubstring indicates whether or not one compound string is contained within another.

string	Specifies the compound string to be searched
substring	Specifies the compound string to be searched for

Return Values

Returns True if *substring* has a single text component and if its text is completely contained within any single text component of *string*; otherwise, it returns False.

Related Information

XmStringCreate(3) and **XmStringCreateLocalized**(3).

XmStringHeight

Purpose

A compound string function that returns the line height of the given compound string

Synopsis

```
#include <Xm/Xm.h>

Dimension XmStringHeight(
        XmRenderTable rendertable,
        XmString string);
```

Description

XmStringHeight returns the height, in pixels, of the sum of all the line heights of the given compound string. Separator components delimit lines.

rendertable	Specifies the render table
string	Specifies the string

Return Values

Returns the height of the specified string.

Related Information

XmStringCreate(3).

XmStringInitContext

Purpose

A compound string function that creates a data structure for scanning an XmString component by component

Synopsis

#include <Xm/Xm.h>

Boolean XmStringInitContext(
    XmStringContext * *context***,**
    XmString *string***);**

Description

XmStringInitContext creates a context to allow applications to read out the contents of a compound string component by component. A Boolean status is returned to indicate that the context could not be initalized.

If the function returns True, the function will allocate space to hold the returned *context*. The application is responsible for managing the allocated space. The memory can be recovered by calling **XmStringFreeContext**.

context	Specifies a pointer to the allocated context
string	Specifies the string

Return Values

Returns True if the context was allocated

Related Information

XmStringCreate(3).

XmStringIsVoid

Purpose

A compound string function that provides information on the existence of non-zero-length text components, tab components, or separator components

Synopsis

#include <Xm/Xm.h>

Boolean XmStringIsVoid(
        XmString ***s1*****);**

Description

XmStringIsVoid returns a Boolean value indicating whether or not string *s1* is void.

s1 Specifies the compound string

Return Values

Returns True if any non-zero-length text components, tab components, or separator components exist in *s1*. That is, the function returns True if the string has no text, tabs, or separators. If *s1* contains the NULL string, the function returns True.

Related Information

XmStringCreate(3).

XmStringLength

Purpose

A compound string function that obtains the length of a compound string

Synopsis

#include <Xm/Xm.h>

```
int XmStringLength(
        XmString s1);
```

Description

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmStringByteStreamLength**. **XmStringLength** obtains the length of a compound string. It returns the number of bytes in *s1* including all tags, direction indicators, and separators. If the compound string has an invalid structure, 0 (zero) is returned.

s1	Specifies the compound string

Return Values

Returns the length of the compound string.

Related Information

XmStringByteStreamLength(3) and **XmStringCreate**(3).

XmStringLineCount

Purpose

A compound string function that returns the number of separators plus one in the provided compound string

Synopsis

#include <Xm/Xm.h>

int XmStringLineCount(
        XmString *string***);**

Description

XmStringLineCount returns the number of separators plus one in the provided compound string. In effect, it counts the lines of text.

string	Specifies the string

Return Values

Returns the number of lines in the compound string. If *string* is empty, the function returns 1. If NULL is passed into *string*, the function returns 0 (zero).

Related Information

XmStringCreate(3).

XmStringNConcat

Purpose

A compound string function that appends a specified number of bytes to a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringNConcat(
    XmString *s1***,**
    XmString *s2***,**
    int *num_bytes***);**

Description

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmStringConcat**. **XmStringNConcat** appends a specified number of bytes from *s2* to the end of *s1*, including tags, directional indicators, and separators. It then returns the resulting compound string. The original strings are preserved. The function allocates space for the resulting compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

s1	Specifies the compound string to which a copy of *s2* is appended.
s2	Specifies the compound string that is appended to the end of *s1*.
num_bytes	Specifies the number of bytes of *s2* to append to *s1*. If this value is less than the length of *s2*, as many bytes as possible, but possibly fewer than this value, will be appended to *s1* such that the resulting string is still a valid compound string.

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3) and **XmStringFree**(3).

XmStringNCopy

Purpose

A compound string function that creates a copy of a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringNCopy(
XmString *s1***,**
int *num_bytes***);**

Description

This function is obsolete and exists for compatibility with previous releases. **XmStringNCopy** creates a copy of *s1* that contains a specified number of bytes, including tags, directional indicators, and separators. It then returns the resulting compound string. The original strings are preserved. The function allocates space for the resulting compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

s1	Specifies the compound string.
num_bytes	Specifies the number of bytes of *s1* to copy. If this value is less than the length of *s1*, as many bytes as possible, but possibly fewer than this value, will be appended to *s1* such that the resulting string is still a valid compound string.

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3) and **XmStringFree**(3).

XmStringParseText

Purpose

A function that converts a character string to a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringParseText(
  XtPointer *text***,**
  XtPointer ****text_end*,**
  XmStringTag *tag***,**
  XmTextType *type***,**
  XmParseTable *parse_table***,**
  Cardinal *parse_count***,**
  XtPointer *call_data***);**

Description

XmStringParseText converts characters specified in *text* to corresponding components in the returned compound string. The resulting compound string consists of at least one locale or charset tag component and a series of **XmString** text components and other components. The conversion proceeds according to the parse information contained in *parse_table*. See the *Motif 2.1—Programmer's Guide* for more information about parsing and parse tables.

- If *type* is **XmCHARSET_TEXT**, the associated *tag* is interpreted as a charset name. If *tag* has a value of NULL, a charset component whose value is the result of mapping **XmFONTLIST_DEFAULT_TAG** is created.
- If *type* is **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**, the associated *tag* is interpreted as a language environment name. If *tag* has a value of NULL, a locale component with a value of **_MOTIF_DEFAULT_LOCALE** is created. If *type* is **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**, *tag* must be NULL or **_MOTIF_DEFAULT_LOCALE**.

XmStringParseText also scans the string for characters that have matches in *parse_table*. Whenever a match is found, the text up to that point is concatenated with the mapped component.

text	Specifies the NULL-terminated string containing characters of a type determined by *type*. This is updated to point to after the last character scanned.
text_end	Specifies a pointer into *text*. If a NULL is supplied to the *text_end* parameter, then **XmStringParseText** parses *text* until NULL is encountered, or until it reaches a point in *text* where it is directed to stop (for example, by a **parse_proc**). Otherwise, the value supplied to the *text_end* parameter is the pointer into *text* where parsing is to stop, and the returned character is the one where parsing did stop.
tag	Specifies the tag to be used in creating the result. The type of string tag created (charset or locale) depends on the text type and the passed in *tag* value. If the *tag* value is NULL and if *type* indicates that a charset string tag should be created, the string tag has the value that is the result of mapping **XmFONTLIST_DEFAULT_TAG**. If *type* indicates a locale string tag, the string tag has the value **_MOTIF_DEFAULT_LOCALE**.
type	Specifies the type of text and the tag type. If a locale tag should be created, *type* has a value of either **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**. If *type* has value of **XmCHARSET_TEXT**, a charset tag will be created.
parse_table	Specifies the parse table to be used in scanning for characters to be converted to other compound string components.
parse_count	Specifies the number of entries in *parse_table*.
call_data	Specifies data to be passed to the parse procedures.

Return Values

Returns a new compound string. The function allocates space to hold the returned compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

Related Information

XmString(3), **XmStringFree**(3), **XmParseTable**(3), **XmParseMapping**(3).

XmStringPeekNextComponent

Purpose

A compound string function that returns the component type of the next component to be fetched

Synopsis

#include <Xm/Xm.h>

XmStringComponentType XmStringPeekNextComponent(
    XmStringContext *context***);**

Description

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmStringPeekNextTriple**. **XmStringPeekNextComponent** examines the next component that would be fetched by **XmStringGetNextComponent** and returns the component type.

context	Specifies the string context structure that was allocated by the **XmStringInitContext** function

Return Values

Returns the type of component found. Refer to the **XmStringComponentType**(3) reference page for a list of component types.

Related Information

XmStringComponentType(3), **XmStringCreate**(3), **XmStringGetNextComponent**(3), and **XmStringInitContext**(3).

XmStringPeekNextTriple

Purpose

A function that returns the component type of the next component

Synopsis

#include <Xm/Xm.h>

XmStringComponentType XmStringPeekNextTriple(
    XmStringContext *context***);**

Description

XmStringPeekNextTriple examines the next component that would be fetched by **XmStringGetNextTriple** and returns the component type.

context	Specifies the string context structure that was allocated by the **XmStringInitContext** function.

Return Values

Returns the type of the component found. Refer to the **XmStringComponentType**(3) reference page for a list of component types.

Related Information

XmString(3), **XmStringComponentType**(3), and **XmStringGetNextTriple**(3).

XmStringPutRendition

Purpose

A convenience function that places renditions around strings

Synopsis

#include <Xm/Xm.h>

XmString XmStringPutRendition(
    XmString *string***,**
    XmStringTag *rendition***);**

Description

XmStringPutRendition places matching **Xm_STRING_COMPONENT_RENDITION_BEGIN** and **XmSTRING_COMPONENT_RENDITION_END** components containing *rendition* around *string*. The original string is preserved.

string Specifies the compound string to which begin and end rendition components should be added.

rendition Specifies the rendition tag to be used in an **XmSTRING_COMPONENT_RENDITION_BEGIN** component which will begin the returned string and in an **XmSTRING_COMPONENT_RENDITION_END** component which will end it.

Return Values

Returns a new compound string. The function allocates space to hold this returned compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

Related Information

XmString(3).

XmStringSegmentCreate

Purpose

A compound string function that creates a compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringSegmentCreate(
    char * *text***,**
    XmStringTag *tag***,**
    XmStringDirection *direction***,**
    Boolean *separator***);**

Description

This function is obsolete and exists for compatibility with previous releases. It can be replaced by using a combination of **XmStringComponentCreate** and **XmStringConcat**. **XmStringSegmentCreate** is a high-level function that assembles a compound string consisting of a font list element tag, a direction component, a text component, and an optional separator component.

The function allocates space for the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmStringFree**.

text	Specifies a NULL-terminated string to be used as the text component of the compound string.
tag	Specifies the tag component to be associated with the text. The value **XmFONTLIST_DEFAULT_TAG** is for compatibility with previous releases.
direction	Specifies the direction of the text.
separator	A value of False means the compound string does not have a separator at the end. A value of True, means a separator immediately follows the text component.

Return Values

Returns a new compound string.

Related Information

XmStringCreate(3).

XmStringSeparatorCreate

Purpose

A compound string function that creates a compound string

Synopsis

#include <Xm/Xm.h>

```
XmString XmStringSeparatorCreate(
 void
```

Description

XmStringSeparatorCreate creates a compound string with a single component, a separator.

Return Values

Returns a new compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

Related Information

XmStringCreate(3).

XmStringTableParseStringArray

Purpose

A convenience function that converts an array of strings to a compound string table

Synopsis

#include <Xm/Xm.h>

XmStringTable XmStringTableParseStringArray(
        XtPointer **strings***,**
        Cardinal *count***,**
        XmStringTag *tag***,**
        XmTextType *type***,**
        XmParseTable *parse***,**
        Cardinal *parse_count***,**
        XtPointer *call_data***);**

Description

XmStringTableParseStringArray takes an array of strings, allocates an **XmStringTable** with an equal number of slots, calls **XmStringParseText** on each string in *strings*, and inserts the resulting **XmString** in the corresponding slot in the **XmStringTable**.

strings	Specifies an array of strings of characters as determined by *type*.
count	Specifies the number of strings in *strings*.
tag	Specifies the tag to be used in creating the result. The type of tag created (charset or locale) depends on the type of the text and the value given. If the value specified is NULL, and *type* indicates that a charset tag should be created, then the tag will have the value of **XmFONTLIST_DEFAULT_TAG**. If *type* indicates a locale tag, then the tag will have the value of **XmFONTLIST_DEFAULT_TAG**.
type	Specifies the type of text to be passed in and the type of tag. If the type is either **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**, a locale tag should be created. If the type is **XmCHARSET_TEXT**, a charset tag will be created.

parse	Specifies the parse table to be used.
parse_count	Specifies the number of entries in the parse table.
call_data	Specifies data to be passed to the parse procedures.

Return Values

Returns a new **XmStringTable**. The function allocates space to hold the **XmStringTable**. When the application no longer needs the returned **XmStringTable**, the application should call **XmStringFree** *count* times (that is, one time for each returned compound string) and then call **XtFree** to deallocate the **XmStringTable** itself.

Related Information

XmStringFree(3) and **XmTabList**(3).

XmStringTableProposeTablist

Purpose

A convenience function that returns a tab list

Synopsis

#include <Xm/Xm.h>

XmTabList XmStringTableProposeTablist(
　　XmStringTable *strings*,
　　Cardinal *num_strings*,
　　Widget *widget*,
　　float *pad_value*,
　　XmOffsetModel *offset_model*);

Description

XmStringTableProposeTablist takes an **XmStringTable** structure containing tabbed compound strings, information on padding between columns, and rendering information and returns a tab list that, if used to render the strings in the table, would cause the strings to line up in columns with no overlap and with the specified amount of padding between the widest item in each column and the start of the next column. Each tab in the tablist would have the same unit type as *units*, an offset model of *offset_model*, and an alignment type of **XmALIGNMENT_BEGINNING**.

strings	Specifies an array of compound strings.
num_strings	Specifies the number of compound strings in *strings*.
widget	Specifies the widget used for deriving any necessary information for creating the rendition. In particular, the **XmNunitType** of *widget* will be used to specify the unit type to be used in determining the amount of padding separating columns and for the tabs in the proposed tab list. Also, *widget*'s render table will be used in interpreting rendition tags within the strings.
pad_value	Specifies the value of the amount of padding to be used to separate columns. The units for this parameter are specified as the **XmNunitType**

set for the *widget* parameter. Refer to the **XmNunitType** resource of the **XmGadget**, **XmManager**, or **XmPrimitive** reference page.

offset_model Specifies the offset model to be used in creating the tabs. Can be **XmABSOLUTE** or **XmRELATIVE**.

Return Values

Returns a new **XmTabList**. The function allocates space to hold the returned tab list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmTabListFree**.

Related Information

XmTabList(3) and **XmTabListFree**(3).

XmStringTableToXmString

Purpose

A convenience function that converts a compound string table to a single compound string

Synopsis

#include <Xm/Xm.h>

XmString XmStringTableToXmString(
    XmStringTable *table***,**
    Cardinal *count***,**
    XmString *break_component***);**

Description

XmStringTableToXmString takes as input *table* of compound strings and a specified string component (such as a tab) and returns a single compound string consisting of each of the elements of *table* concatenated together with a single copy of *break_component* inserted between each element.

table Specifies an **XmStringTable** containing the compound strings to be converted.

count Specifies the number of compound strings in *table*.

break_component
Specifies the *XmStringComponent* that will be inserted in the result to separate the individual elements of *table*. The most useful types will be **XmSTRING_COMPONENT_SEPARATOR** and **XmSTRING_COMPONENT_TAB**. Refer to the **XmStringComponentType**(3) reference page for a complete list of possible component types. Note, however, that the **XmSTRING_COMPONENT_UNKNOWN** component is not a possible type.

Return Values

Returns a new **XmString**. The function will allocate space to hold the returned compound string. When the application no longer needs the returned compound string, the application should call **XmStringFree**.

Related Information

XmString(3), **XmStringComponentType**(3), and **XmStringFree**(3).

XmStringTableUnparse

Purpose

A convenience function that converts a table of compound strings to an array of text

Synopsis

#include <Xm/Xm.h>

XtPointer * XmStringTableUnparse(
XmStringTable *table***,**
Cardinal *count***,**
XmStringTag *tag***,**
XmTextType *tag_type***,**
XmTextType *output_type***,**
XmParseTable *parse***,**
Cardinal *parse_count***,**
XmParseModel *parse_model***);**

Description

XmStringTableUnparse takes an array of compound strings, allocates a string array for the type of characters determined by *type* with an equal number of slots, calls **XmStringUnparse** on each compound string in *table*, and inserts the resulting string in the corresponding slot in the array.

table	Specifies an **XmStringTable** containing the compound string to be converted.
count	Specifies the number of compound strings in *table*.
tag	Specifies the tag to be used in matching with text segments. The two types of tag types are **XmFONTLIST_DEFAULT_TAG** and **_MOTIF_DEFAULT_LOCALE**. Only segments tagged with *tag* will be returned. If *tag* is NULL, all segments will be matched.
tag_type	Specifies the type of tag to be searched for. These types include **XmMULTIBYTE_TEXT**, **XmWIDECHAR_TEXT**, and **XmCHARSET_TEXT**.

output_type Specifies the type of text to be generated. These types include **XmMULTIBYTE_TEXT**, **XmWIDECHAR_TEXT**, and **XmCHARSET_TEXT**.

parse Specifies the parse table to be used.

parse_count Specifies the number of items in *parse*.

parse_model Specifies which non-text components to be considered in matching in *parse_table*. Possible values are:

XmOUTPUT_ALL
Puts out all matching components.

XmOUTPUT_BETWEEN
Puts out only those matching components that are between two matching text components.

XmOUTPUT_BEGINNING
Puts out only those matching components that are at the beginning of a matching text component.

XmOUTPUT_END
Puts out only those matching components that are at the end of a matching text component.

XmOUTPUT_BOTH
Puts out only those matching components that are at the beginning or end of a matching text component.

Return Values

Returns an allocated array of allocated strings. The application is responsible for managing the allocated space. The application can recover the allocated strings space by calling **XtFree** *count* times (that is, one time for each allocated string). The application can then recover the allocated array by calling **XtFree** on the allocated array itself.

Related Information

XmStringTab.

XmStringToXmStringTable

Purpose

A convenience function that converts a single compound string to a table of compound strings

Synopsis

#include <Xm/Xm.h>

Cardinal XmStringToXmStringTable(
    XmString *string***,**
    XmString *break_component***,**
    XmStringTable **table***);**

Description

XmStringToXmStringTable takes as input a single compound string and a specified string component (such as a tab) and returns a table of compound strings consisting of portions of *string* delimited by components matching *break_component*. The components marking breaks will not appear in the resulting table.

string Specifies the **XmString** to be converted.

break_component
Specifies the *XmStringComponent* that will be used to indicate where to split *string* to form the individual elements of *table*. The most useful types will be **XmSTRING_COMPONENT_SEPARATOR** and **XmSTRING_COMPONENT_TAB**. Refer to the **XmStringComponentType**(3) reference page for a complete list of possible component types. Note, however, that the **XmSTRING_COMPONENT_UNKNOWN** component is not a possible type.

table Returns the equivalent **XmStringTable**. The function will allocate space to hold the returned **XmStringTable**. When the applicaiton no longer needs the returned **XmStringTable**, the application should call **XmStringFree** once for each compound string in the table, and then calling **XtFree** to deallocate the **XmStringTable** itself.

Return Values

Returns the number of compound strings in *table*.

Related Information

XmStringTable(3).

XmStringUnparse

Purpose

A compound string function that unparses text

Synopsis

#include <Xm/Xm.h>

XtPointer XmStringUnparse(
    XmString *string*,
    XmStringTag *tag*,
    XmTextType *tag_type*,
    XmTextType *output_type*,
    XmParseTable *parse_table*,
    Cardinal *parse_count*,
    XmParseModel *parse_model*);

Description

XmStringUnparse looks in the input *string* for text segments that are tagged with locale or charset tags that match *tag*. The *tag_type* parameter specifies whether the tag is a locale or charset type. If *tag* has a value of NULL, all the segments are matched. When a text segment is found with a matching tag, it is added to the end of a resulting string. The characters in the resulting string are of type *output_type*.

XmStringUnparse also checks *string* for components that match components in *parse_table*, and also to see if the component matches the condition specified by *parse_model*. If the string component matches in both checks, then the associated character is added to the end of the resulting string.

string	Specifies the **XmString** to be converted.
tag	Specifies the tag to be used in matching with text segments. Only text segments that match *tag* will be included in the resulting string. If *tag* has a value of NULL, all segments are considered as matches, and *tag_type* is ignored.

tag_type Specifies the type of tag to be searched for, including **XmMULTIBYTE_TEXT**, **XmWIDECHAR_TEXT**, and **XmCHARSET_TEXT**.

output_type Specifies the type of text to be returned in the string, including **XmMULTIBYTE_TEXT**, **XmWIDECHAR_TEXT**, and **XmCHARSET_TEXT**.

parse_table Specifies the parse table to be used in scanning for compound string components to be converted to other characters.

parse_count Specifies how many entries are in *parse_table*.

parse_model Specifies which non-text components to be considered in matching in *parse_table*. These include:

XmOUTPUT_ALL
Puts out all matching components.

XmOUTPUT_BETWEEN
Puts out only those matching components that are between two matching text components.

XmOUTPUT_BEGINNING
Puts out only those matching components that are at the beginning of a matching text component.

XmOUTPUT_END
Puts out only those matching components that are at the end of a matching text component.

XmOUTPUT_BOTH
Puts out only those matching components that are at the beginning or end of a matching text component.

Return Values

Returns a newly allocated string containing characters of a type determined by *output_type*. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XtFree**.

Related Information

XmString(3), **XmParseTable**(3), **XmParseMapping**(3).

XmStringWidth

Purpose

A compound string function that returns the width of the widest line in a compound string

Synopsis

#include <Xm/Xm.h>

Dimension XmStringWidth(
XmRenderTable *rendertable***,**
XmString *string***);**

Description

XmStringWidth returns the width, in pixels, of the widest line in the provided compound string.

rendertable	Specifies the render table
string	Specifies the string

Return Values

Returns the width of the compound string.

Related Information

XmStringCreate(3).

XmTabCreate

Purpose

A convenience function that creates a tab stop

Synopsis

#include <Xm/Xm.h>

XmTab XmTabCreate(
float *value***,**
unsigned char *units***,**
XmOffsetModel *offset_model***,**
unsigned char *alignment***,**
char **decimal***);**

Description

XmTabCreate creates a tab stop at a position defined by the *value* and *units* arguments.

value	Specifies the floating point value to be used in conjunction with *units* to calculate the location of the tab stop. Note that negative values are not permitted.
units	Specifies the unit type (for example, **XmMILLIMETERS**) to be used in conjunction with *value* to calculate the location of the tab stop. You can specify any unit described by the **XmConvertUnits** reference page. For resources of type, dimension, or position, you can specify units as described in the **XmNunitType** resource of the **XmGadget**, **XmManager**, or **XmPrimitive** reference page.
offset_model	Specifies whether the tab value represents an absolute position or a relative offset from the previous tab. Valid values are **XmABSOLUTE** and **XmRELATIVE**.
alignment	Specifies how the text should be aligned relative to this tab stop. Valid values are **XmALIGNMENT_BEGINNING**.

decimal Specifies the multibyte character in the current language environment to be used as the decimal point for a decimal aligned tab stop. This is currently unused.

Return Values

Returns a newly allocated **XmTab**. The application is responsible for managing this allocated space. The application can recover this allocated space by calling **XmTabFree**.

Related Information

XmTab(3) and **XmTabFree**(3).

XmTabFree

Purpose

A convenience function that frees a tab

Synopsis

```
#include <Xm/Xm.h>

void XmTabFree(
        XmTab tab);
```

Description

XmTabFree frees the memory associated with the specified tab.

tab	Specifies the tab to be freed.

Related Information

XmTab(3).

XmTabGetValues

Purpose

A convenience function that returns tab values

Synopsis

#include <Xm/Xm.h>

float XmTabGetValues(
    XmTab *tab***,**
    unsigned char **units***,**
    XmOffsetModel **offset***,**
    unsigned char **alignment***,**
    char ***decimal***);**

Description

XmTabGetValues takes an **XmTab** structure, returns the floating point number that is set as the value of the tab, and then sets values for the *units*, *offset*, *alignment*, and *decimal* arguments where they are not NULL. The returned floating point number represents the distance that the rendering of the **XmString** segment associated with *tab* will be offset. The offset is from either the beginning of the rendering or from the previous tab stop, depending on the setting for the *offset* model. The distance will use the unit type pointed at by *unit*.

tab	Specifies the tab to get the value from.
units	Specifies a pointer to the unit type.
offset	Specifies a pointer to the offset model.
alignment	Specifies a pointer to the alignment type.
decimal	Specifies a pointer to the multibyte character used as the decimal point.

Return Values

Returns a floating point number that is set as the value of the tab.

Related Information

XmTab(3).

XmTabListCopy

Purpose

A convenience function that creates a new tab list from an existing list

Synopsis

#include <Xm/Xm.h>

XmTabList XmTabListCopy(
XmTabList *tablist***,**
int *offset***,**
Cardinal *count***);**

Description

XmTabListCopy creates a new tab list consisting of a copy of a portion of the contents of the *tablist* argument. This function starts copying at the specified offset value of the tab list and copies *count* values.

tablist	Specifies a tab list to be copied.
offset	Specifies where to start copying. A value of 0 (zero) indicates begin at the beginning, a value of 1 indicates to skip the first tab, and so on. A negative indicates to begin counting backwards from the end. A value of -1 indicates to start copying from the last tab.
count	Specifies the number of tabs to copy. A value of 0 (zero) indicates to copy all elements from the starting point to the end (beginning if *offset* is negative) of the tab list.

Return Values

If *tablist* is NULL, this function returns NULL. Otherwise, this function returns a newly allocated **XmTabList**. If the function does allocate an **XmTabList**, then the application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmTabListFree**.

Related Information

XmTabList(3) and **XmTabListFree**(3).

XmTabListFree

Purpose

A convenience function that frees the memory of a new tab list

Synopsis

#include <Xm/Xm.h>

void XmTabListFree(
    XmTabList *tablist***);**

Description

XmTabListFree recovers memory used by a tab list. In addition, this function frees all contained tabs. If the *tablist* is NULL, the function returns immediately.

tablist	Specifies the tab list to be freed.

Related Information

XmTabList(3).

XmTabListGetTab

Purpose

A convenience function that returns a copy of a tab

Synopsis

#include <Xm/Xm.h>

XmTab XmTabListGetTab(
    XmTabList *tablist***,**
    Cardinal *position***);**

Description

XmTabListGetTab returns a copy of the tab that is located at the specified position in the tab list.

tablist	Specifies the tab list.
position	Specifies the position of the tab to be returned. A value of 0 (zero) returns the first tab in the tab list, a value of 1 returns the second tab, and so on.

Return Values

Returns a copy of the tab that is located at the specified position in the tab list. If *position* is greater than or equal to the number of tabs in the tab list, this function returns NULL. The application is responsible for managing the space allocted by the returned tab copy. The application can recover this allocated space by calling **XmTabFree**.

Related Information

XmTabFree(3) and **XmTabList**(3).

XmTabListInsertTabs

Purpose

A convenience function that inserts tabs into a tab list

Synopsis

#include <Xm/Xm.h>

XmTabList XmTabListInsertTabs(
  XmTabList *oldlist***,**
  XmTab **tabs***,**
  Cardinal *tab_count***,**
  int *position***);**

Description

XmTabListInsertTabs creates a new tab list that includes the tabs in *oldlist*. This function copies specified tabs to the tab list at the given position. The first *tab_count* tabs of the *tabs* array are added to the tab list. If *oldlist* is NULL, **XmTabListInsertTabs** creates a new tab list containing only the tabs specified.

oldlist Specifies the tab list to add the tabs to. The function deallocates *oldlist* after extracting the required information.

tabs Specifies a pointer to the tabs to be added to the tab list. It is the caller's responsibility to free the tabs in *tabs* by using **XmTabFree**.

tab_count Specifies the number of tabs in *tabs*.

position Specifies the position of the first new tab in the tab list. A value of 0 (zero) makes the first new tab the first tab in the tab list, a value of 1 makes it the second tab, and so on. If *position* is greater than the number of tabs in *oldlist*, then the tabs will be inserted at the end. If *position* is negative, the count will be backwards from the end. A value of -1 makes the first new tab the last tab, and so on.

Return Values

If *tabs* is NULL or *tab_count* is 0 (zero), this function returns *oldlist*. Otherwise, it returns a new tab list. The function allocates space to hold the returned tab list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmTabListFree**.

Related Information

XmTabList(3) and **XmTabListFree**(3).

XmTabListRemoveTabs

Purpose

A convenience function that removes noncontiguous tabs

Synopsis

#include <Xm/Xm.h>

XmTabList XmTabListRemoveTabs(
  XmTabList *oldlist***,**
  Cardinal **position_list***,**
  Cardinal *position_count***);**

Description

XmTabListRemoveTabs removes noncontiguous tabs from a tab list. The function creates a new tab list by copying the contents of *oldlist* and removing all tabs whose corresponding positions appear in the *position_list* array. A warning message is displayed if a specified position is invalid; for example, if the value is a number greater than the number of tabs in the tab list.

tablist Specifies the tab list. The function deallocates *oldlist* and the tabs it contains after extracting the required information.

position_list Specifies an array of the tab positions to be removed. The position of the first tab in the list is 0 (zero), the position of the second tab is 1, and so on.

position_count
Specifies the number of elements in the *position_list*.

Return Values

If *oldlist* or *position_list* is NULL, or *position_count* is 0 (zero), returns *oldlist*. Otherwise, this function returns the new tab list. The function allocates space to hold the returned tab list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmTabListFree**.

Related Information

XmTabList(3) and **XmTabListFree**(3).

XmTabListReplacePositions

Purpose

A convenience function that creates a new tab list with replacement tabs

Synopsis

#include <Xm/Xm.h>

XmTabList XmTabListReplacePositions(
    XmTabList *oldlist***,**
    Cardinal **position_list***,**
    XmTab **tabs***,**
    Cardinal *tab_count***);**

Description

XmTabListReplacePositions creates a new tab list that contains the contents of *oldlist*, but with the tabs at the positions in *position_list* replaced with copies of the corresponding tabs in *tabs*. A warning message is displayed if a specified position is invalid; for example, if the value is a number greater than the number of tabs in the tab list.

This function deallocates the original tab list after extracting the required information. It is the caller's responsibility to free the tabs in *tabs* by using the **XmTabFree** function.

oldlist	Specifies the tab list. The function deallocates the tab list after extracting the required information.
position_list	Specifies an array of positions of the tabs to be replaced. The position of the first tab is 0 (zero), the position of the second tab is 1, and so on.
tabs	Specifies an array of the replacement tabs.
tab_count	Specifies the number of elements in *position_list* and *tabs*.

Return Values

If *tabs*, *oldlist*, or *position_list* is NULL, or *tab_count* is 0 (zero), returns *oldlist*. Otherwise, this function returns the new tab list. The function allocates space to hold the returned tab list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling **XmTabListFree**.

Related Information

XmTabList(3).

XmTabListTabCount

Purpose

A convenience function that counts the number of tabs

Synopsis

```
#include <Xm/Xm.h>

Cardinal XmTabListTabCount(
        XmTabList tablist);
```

Description

XmTabListTabCount counts the number of tabs in the specified *tablist*.

tablist	Specifies the tab list.

Return Values

Returns the number of tabs in *tablist*.

Related Information

XmTabList(3).

XmTabSetValue

Purpose

A convenience function that sets a tab stop

Synopsis

#include <Xm/Xm.h>

void XmTabSetValue(
    XmTab *tab***,**
    float *value***);**

Description

XmTabSetValue sets the *value* field of the **XmTab** structure associated with *tab*.

tab	Specifies the tab to set the value of.
value	Specifies the floating point number which represents the distance that the rendering of the **XmString** segment associated with *tab* will be offset. The offset is from either the beginning of the rendering or from the previous tab stop, depending on the setting for the *offset* model. The distance depends on the tab’s unit type. Note that negative values are not permitted, and that if a tab stop would cause text to overlap, the x position for the segment is set immediately after the end of the previous segment.

Related Information

See also the *Motif 2.1—Programmer’s Guide* for more information about tabs and tab lists. **XmTab**(3).

XmTargetsAreCompatible

Purpose

A function that tests whether the target types match between a drop site and source object

Synopsis

#include <Xm/DragDrop.h>

Boolean XmTargetsAreCompatible(
Display ****display*,**
Atom ****export_targets*,**
Cardinal *num_export_targets***,**
Atom ****import_targets*,**
Cardinal *num_import_targets***);**

Description

XmTargetsAreCompatible determines whether the import targets of the destination match any of the export targets of a source. If there is at least one target in common, the function returns True.

display Specifies the display connection.

export_targets
Specifies the list of target atoms associated with the source object. This resource identifies the selection targets the source can convert to.

num_export_targets
Specifies the number of entries in the list of export targets.

import_targets
Specifies the list of targets to be checked against the **XmNexportTargets** of the source associated with the specified DragContext

num_import_targets
Specifies the number of entries in the *import_targets* list.

Return Values

Returns a Boolean value that indicates whether the destination targets are compatible with the source targets. If there is at least one target in common, the routine returns True; otherwise, returns False.

Related Information

XmDragContext(3) and **XmDropSite**(3).

XmTextClearSelection

Purpose

A Text function that clears the primary selection

Synopsis

```
#include <Xm/Text.h>

void XmTextClearSelection(
        Widget widget,
        Time time);
```

Description

XmTextClearSelection clears the primary selection in the Text widget.

widget	Specifies the Text widget ID.
time	Specifies the server time at which the selection value is desired. This should be the time of the event that triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed()**.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextCopy

Purpose

A Text function that copies the primary selection to the clipboard

Synopsis

#include <Xm/Text.h>

Boolean XmTextCopy(
Widget *widget***,**
Time *time***);**

Description

XmTextCopy copies the primary selected text to the clipboard.

This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmCOPY**.

widget	Specifies the Text widget ID.
time	Specifies the server time at which the selection value is to be modified. This should be the time of the event which triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed()**.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

Related Information

XmText(3).

XmTextCopyLink

Purpose

A Text function that copies a link to the primary selection to the clipboard

Synopsis

#include <Xm/Text.h>

Boolean XmTextCopyLink(
  Widget *widget***,**
  Time *time***);**

Description

XmTextCopyLink copies a link to the primary selected text to the clipboard. This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmLINK**. The Text widget itself does not copy any links; **XmNconvertCallback** procedures are responsible for copying the link to the clipboard and for taking any related actions.

widget	Specifies the Text widget ID.
time	Specifies the time of the transfer. This should be the time of the event which triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed**.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

Related Information

XmText(3).

XmTextCut

Purpose

A Text function that copies the primary selection to the clipboard and deletes the selected text

Synopsis

```
#include <Xm/Text.h>

Boolean XmTextCut(
        Widget widget,
        Time time);
```

Description

XmTextCut copies the primary selected text to the clipboard and then deletes the primary selected text. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmMOVE**. If the transfer is successful, this routine then calls the **XmNconvertCallback** procedures for the *CLIPBOARD* selection and the *DELETE* target.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is to be modified. This should be the time of the event that triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed()**.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

Related Information

XmText(3).

XmTextDisableRedisplay

Purpose

A Text function that temporarily prevents visual update of the Text widget

Synopsis

```
#include <Xm/Text.h>

void XmTextDisableRedisplay(
        Widget widget);
```

Description

XmTextDisableRedisplay prevents redisplay of the specified Text widget even though its visual attributes have been modified. The visual appearance of the widget remains unchanged until **XmTextEnableRedisplay** is called, although the insertion cursor is not displayed. This allows an application to make multiple changes to the widget without causing intermediate visual updates.

widget	Specifies the Text widget ID

Related Information

XmTextEnableRedisplay(3).

XmTextEnableRedisplay

Purpose

A Text function that forces the visual update of a Text widget

Synopsis

#include <Xm/Text.h>

void XmTextEnableRedisplay(
    Widget *widget***);**

Description

XmTextEnableRedisplay is used in conjunction with **XmTextDisableRedisplay**, which suppresses visual update of the Text widget. When **XmTextEnableRedisplay** is called, it determines if any visual attributes have been set or modified for the specified widget since **XmTextDisableRedisplay** was called. If so, it forces the widget to update its visual display for all of the intervening changes. Any subsequent changes that affect visual appearance cause the widget to update its visual display. This function also causes the insertion cursor, which is not shown while redisplay is disabled, to be restored.

widget	Specifies the Text widget ID

Related Information

XmTextDisableRedisplay(3).

XmTextFieldClearSelection

Purpose

A TextField function that clears the primary selection

Synopsis

#include <Xm/TextF.h>

void XmTextFieldClearSelection(
    Widget *widget***,**
    Time *time***);**

Description

XmTextFieldClearSelection clears the primary selection in the TextField widget.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is desired. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3).

XmTextFieldCopy

Purpose

A TextField function that copies the primary selection to the clipboard

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldCopy(
    Widget *widget***,**
    Time *time***);**

Description

XmTextFieldCopy copies the primary selected text to the clipboard.

This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmCOPY**.

widget	Specifies the TextField widget ID.
time	Specifies the time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

Related Information

XmTextField(3).

XmTextFieldCopyLink

Purpose

A TextField function that copies a link to the primary selection to the clipboard

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldCopyLink(
 Widget *widget***,**
 Time *time***);**

Description

XmTextFieldCopyLink copies a link to the primary selected text to the clipboard. This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmLINK**. The TextField widget itself does not copy any links; **XmNconvertCallback** procedures are responsible for copying the link to the clipboard and for taking any related actions.

widget	Specifies the TextField widget ID.
time	Specifies the time of the transfer. This should be the time of the event which triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed**.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

Related Information

XmTextField(3).

XmTextFieldCut

Purpose

A TextField function that copies the primary selection to the clipboard and deletes the selected text

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldCut(
Widget *widget***,**
Time *time***);**

Description

XmTextFieldCut copies the primary selected text to the clipboard and then deletes the primary selected text. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

This routine calls the **XmNconvertCallback** procedures, possibly multiple times, with the *selection* member of the **XmConvertCallbackStruct** set to *CLIPBOARD* and with the *parm* member set to **XmMOVE**. If the transfer is successful, this routine then calls the **XmNconvertCallback** procedures for the *CLIPBOARD* selection and the *DELETE* target.

widget	Specifies the TextField widget ID.
time	Specifies the time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

Related Information

XmTextField(3).

XmTextFieldGetBaseline

Purpose

A TextField function that accesses the y position of the baseline

Synopsis

#include <Xm/TextF.h>

int XmTextFieldGetBaseline(
        Widget *widget***);**

Description

XmTextFieldGetBaseline accesses the *y* position of the baseline in the TextField widget, relative to the *y* position of the top of the widget.

widget	Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns an integer value that indicates the *y* position of the baseline in the TextField widget. The calculation takes into account the margin height, shadow thickness, highlight thickness, and font ascent of the first font (set) in the fontlist used for drawing text. In this calculation, the *y* position of the top of the widget is 0 (zero).

Related Information

XmTextField(3).

XmTextFieldGetEditable

Purpose

A TextField function that accesses the edit permission state

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldGetEditable(
    Widget *widget***);**

Description

XmTextFieldGetEditable accesses the edit permission state of the TextField widget.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns a Boolean value that indicates the state of the **XmNeditable** resource.

Related Information

XmTextField(3).

XmTextFieldGetInsertionPosition

Purpose

A TextField function that accesses the position of the insertion cursor

Synopsis

```
#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetInsertionPosition(
        Widget widget);
```

Description

XmTextFieldGetInsertionPosition accesses the insertion cursor position of the TextField widget.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns an **XmTextPosition** value that indicates the state of the **XmNcursorPosition** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

Related Information

XmTextField(3).

XmTextFieldGetLastPosition

Purpose

A TextField function that accesses the position of the last text character

Synopsis

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetLastPosition(
    Widget *widget***);**

Description

XmTextFieldGetLastPosition accesses the position of the last character in the text buffer of the TextField widget.

widget	Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns an **XmTextPosition** value that indicates the position of the last character in the text buffer. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). The last character position is equal to the number of characters in the text buffer.

Related Information

XmTextField(3).

XmTextFieldGetMaxLength

Purpose

A TextField function that accesses the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

#include <Xm/TextF.h>

int XmTextFieldGetMaxLength(
    Widget *widget***);**

Description

XmTextFieldGetMaxLength accesses the value of the current maximum allowable length of the text string in the TextField widget entered from the keyboard. The maximum allowable length prevents the user from entering a text string larger than this limit. Note that the maximum allowable length is the same as the value of the widget’s **XmNmaxLength** resource.

widget	Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns the integer value that indicates the string’s maximum allowable length that can be entered from the keyboard.

Related Information

XmTextField(3).

XmTextFieldGetSelection

Purpose

A TextField function that retrieves the value of the primary selection

Synopsis

#include <Xm/TextF.h>

char * XmTextFieldGetSelection(
  Widget *widget***);**

Description

XmTextFieldGetSelection retrieves the value of the primary selection. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget	Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns a character pointer to the string that is associated with the primary selection.

Related Information

XmTextField(3) and **XmTextFieldGetSelectionWcs**(3).

XmTextFieldGetSelectionPosition

Purpose

A TextField function that accesses the position of the primary selection

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldGetSelectionPosition(
    Widget *widget***,**
    XmTextPosition **left***,**
    XmTextPosition **right***);**

Description

XmTextFieldGetSelectionPosition accesses the left and right position of the primary selection in the text buffer of the TextField widget.

widget	Specifies the TextField widget ID
left	Specifies the pointer in which the position of the left boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).
right	Specifies the pointer in which the position of the right boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns True if the widget owns the primary selection; otherwise, it returns False.

Related Information

XmTextField(3).

XmTextFieldGetSelectionWcs

Purpose

A TextField function that retrieves the value of a wide character encoded primary selection

Synopsis

#include <Xm/TextF.h>

wchar_t * XmTextFieldGetSelectionWcs(
        Widget *widget***);**

Description

XmTextFieldGetSelectionWcs retrieves the value of the primary selection, encoded in a wide character format. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the wide character buffer by calling **XtFree**.

widget	Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns the wide character string that is associated with the primary selection in the TextField widget.

Related Information

XmTextField(3) and **XmTextFieldGetSelection**(3).

XmTextFieldGetString

Purpose

A TextField function that accesses the string value

Synopsis

```
#include <Xm/TextF.h>

char * XmTextFieldGetString(
        Widget widget);
```

Description

XmTextFieldGetString accesses the string value of the TextField widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns a character pointer to the string value of the TextField widget. This returned value is a copy of the value of the **XmNvalue** resource. Returns an empty string if the length of the TextField widget's string is 0 (zero).

Related Information

XmTextField(3) and **XmTextFieldGetStringWcs**(3).

XmTextFieldGetStringWcs

Purpose

A TextField function that retrieves a copy of the wide character string value of a TextField widget

Synopsis

#include <Xm/TextF.h>

wchar_t * XmTextFieldGetStringWcs(
  Widget *widget***);**

Description

XmTextFieldGetStringWcs retrieves a copy of the wide character string value of the TextField widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget	Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns the wide character string value of the TextField widget. The function returns an empty string if the length of the TextField widget's string is 0 (zero).

Related Information

XmTextField(3) and **XmTextFieldGetString**(3).

XmTextFieldGetSubstring

Purpose

A TextField function that retrieves a copy of a portion of the internal text buffer

Synopsis

#include <Xm/TextF.h>

int XmTextFieldGetSubstring(
  Widget *widget***,**
  XmTextPosition *start***,**
  int *num_chars***,**
  int *buffer_size***,**
  char ****buffer*);**

Description

XmTextFieldGetSubstring retrieves a copy of a portion of the internal text buffer of a TextField widget. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

The size of the required buffer depends on the maximum number of bytes per character (**MB_CUR_MAX**) for the current locale. **MB_CUR_MAX** is a macro defined in **stdlib.h**. The buffer should be large enough to contain the substring to be copied and a NULL terminator. Use the following equation to calculate the size of buffer the application should provide:

buffer_size **= (***num_chars**** MB_CUR_MAX) + 1**

widget	Specifies the TextField widget ID.
start	Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
num_chars	Specifies the number of characters to be copied into the provided buffer.

buffer_size Specifies the size of the supplied buffer in bytes. This size should account for a NULL terminator.

buffer Specifies the character buffer into which the internal text buffer will be copied.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmTextField(3) and **XmTextFieldGetSubstringWcs**(3).

XmTextFieldGetSubstringWcs

Purpose

A TextField function that retrieves a portion of a wide character internal text buffer

Synopsis

#include <Xm/TextF.h>

int XmTextFieldGetSubstringWcs(
  Widget *widget*,
  XmTextPosition *start*,
  int *num_chars*,
  int *buffer_size*,
  wchar_t ****buffer*);**

Description

XmTextFieldGetSubstringWcs retrieves a copy of a portion of the internal text buffer of a TextField widget that is stored in a wide character format. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

widget	Specifies the TextField widget ID.
start	Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
num_chars	Specifies the number of **wchar_t** characters to be copied into the provided buffer.
buffer_size	Specifies the size of the supplied buffer as a number of **wchar_t** storage locations. The minimum size is *num_chars* + 1.
buffer	Specifies the wide character buffer into which the internal text buffer will be copied.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

XmCOPY_SUCCEEDED
: The function was successful.

XmCOPY_FAILED
: The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
: The requested number of characters extended beyond the internal buffer. The function copied characters to the end of the buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmTextField(3) and **XmTextFieldGetSubstring**(3).

XmTextFieldInsert

Purpose

A TextField function that inserts a character string into a text string

Synopsis

#include <Xm/TextF.h>

void XmTextFieldInsert(
  Widget *widget*,
  XmTextPosition *position*,
  char * *value***);**

Description

XmTextFieldInsert inserts a character string into the text string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. If the **XmNcursorPosition** resource is greater than or is the same value as *position*, the **XmNmotionVerifyCallback** is called.

widget	Specifies the TextField widget ID
position	Specifies the position in the text string where the character string is to be inserted
value	Specifies the character string value to be added to the text widget

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3) and **XmTextFieldInsertWcs**(3).

XmTextFieldInsertWcs

Purpose

A TextField function that inserts a wide character string into a TextField widget

Synopsis

#include <Xm/TextF.h>

void XmTextFieldInsertWcs(
    Widget *widget***,**
    XmTextPosition *position***,**
    wchar_t ****wcstring*);**

Description

XmTextFieldInsertWcs inserts a wide character string into the TextField widget at a specified location. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. If the **XmNcursorPosition** resource is greater than or is the same value as *position*, the **XmNmotionVerifyCallback** is called.

widget	Specifies the TextField widget ID
position	Specifies the position in the text string where the new character string is to be inserted
wcstring	Specifies the wide character string value to be added to the TextField widget

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3) and **XmTextFieldInsert**(3).

XmTextFieldPaste

Purpose

A TextField function that inserts the clipboard selection

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldPaste(
Widget *widget***);**

Description

XmTextFieldPaste inserts the clipboard selection at the insertion cursor of the destination widget. If **XmNpendingDelete** is True and the insertion cursor is inside the current selection, the clipboard selection replaces the selected text.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

This routine calls the widget's **XmNdestinationCallback** procedures with the *selection* member of the **XmDestinationCallbackStruct** set to *CLIPBOARD* and with the *operation* member set to **XmCOPY**. If the **XmNcursorPosition** resource is greater than or is the same value as the position where the selection is to be inserted, the **XmNmotionVerifyCallback** is called.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns False if no transfers take place. Otherwise, it returns True.

Related Information

XmTextField(3).

XmTextFieldPasteLink

Purpose

A TextField function that inserts a link to the clipboard selection

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldPasteLink(
    Widget *widget***);**

Description

XmTextFieldPasteLink inserts a link to the clipboard selection at the insertion cursor. This routine calls the widget's **XmNdestinationCallback** procedures with the *selection* member of the **XmDestinationCallbackStruct** set to *CLIPBOARD* and with the *operation* member set to **XmLINK**. The TextField widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the clipboard selection and for taking any related actions.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns False if no transfers take place. Otherwise, it returns True.

Related Information

XmTextField(3).

XmTextFieldPosToXY

Purpose

A TextField function that accesses the x and y position of a character position

Synopsis

#include <Xm/TextF.h>

Boolean XmTextFieldPosToXY(
Widget *widget*,
XmTextPosition *position*,
Position **x*,
Position **y*);

Description

XmTextFieldPosToXY accesses the *x* and *y* position, relative to the upper left corner of the TextField widget, of a given character position in the text buffer.

widget	Specifies the TextField widget ID
position	Specifies the character position in the text for which the *x* and *y* position is accessed. This is an integer number of characters from the beginning of the buffer. The first character position is 0.
x	Specifies the pointer in which the *x* position is returned. The returned position is the distance from the left side of the widget to the left border of the character. This value is meaningful only if the function returns True.
y	Specifies the pointer in which the *y* position is returned. The returned position is the distance from the top of the widget to the character's baseline. This value is meaningful only if the function returns True.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns True if the character position is displayed in the TextField widget; otherwise, it returns False, and no *x* or *y* value is returned.

Related Information

XmTextField(3).

XmTextFieldRemove

Purpose

A TextField function that deletes the primary selection

Synopsis

```
#include <Xm/TextF.h>

Boolean XmTextFieldRemove(
        Widget widget);
```

Description

XmTextFieldRemove deletes the primary selected text. If there is a selection, this routine also calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This function may also call the **XmNmotionVerifyCallback** callback.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

This function returns False if the primary selection is NULL or if the *widget* does not own the primary selection. Otherwise, it returns True.

Related Information

XmTextField(3).

XmTextFieldReplace

Purpose

A TextField function that replaces part of a text string

Synopsis

#include <Xm/TextF.h>

void XmTextFieldReplace(
Widget *widget***,**
XmTextPosition *from_pos***,**
XmTextPosition *to_pos***,**
char * *value***);**

Description

XmTextFieldReplace replaces part of the text string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both parameters, *from_pos* and *to_pos*, must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. The **XmNmotionVerifyCallback** is generated if *to_pos* is less than or equal to the cursor position and the length of *value* is not the same as the length of the text being replaced, or if the cursor position is between *from_pos* and *to_pos*, and the distance from the cursor position to *from_pos* is greater than the length of *value*.

widget	Specifies the TextField widget ID
from_pos	Specifies the start position of the text to be replaced

to_pos	Specifies the end position of the text to be replaced
value	Specifies the character string value to be added to the text widget

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3). **XmTextFieldReplaceWcs**(3).

XmTextFieldReplaceWcs

Purpose

A TextField function that replaces part of a wide character string in a TextField widget

Synopsis

#include <Xm/TextF.h>

void XmTextFieldReplaceWcs(
  Widget *widget***,**
  XmTextPosition *from_pos***,**
  XmTextPosition *to_pos***,**
  wchar_t ****wcstring*);**

Description

XmTextFieldReplaceWcs replaces part of the wide character string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both parameters, *from_pos* and *to_pos*, must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. If the **XmNcursorPosition** resource is greater than or is the same value as *from_pos*, the **XmNmotionVerifyCallback** is called.

widget	Specifies the TextField widget ID
from_pos	Specifies the start position of the text to be replaced
to_pos	Specifies the end position of the text to be replaced

wcstring	Specifies the wide character string value to be added to the TextField widget

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3) and **XmTextFieldReplace**(3).

XmTextFieldSetAddMode

Purpose

A TextField function that sets the state of Add mode

Synopsis

#include <Xm/TextF.h>

void XmTextFieldSetAddMode(
    Widget *widget***,**
    Boolean *state***);**

Description

XmTextFieldSetAddMode controls whether or not the TextField widget is in Add mode. When the widget is in Add mode, the insert cursor can be moved without disturbing the primary selection.

widget	Specifies the TextField widget ID
state	Specifies whether or not the widget is in Add mode. A value of True turns on Add mode; a value of False turns off Add mode.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3).

XmTextFieldSetEditable

Purpose

A TextField function that sets the edit permission

Synopsis

#include <Xm/TextF.h>

void XmTextFieldSetEditable(
Widget *widget***,**
Boolean *editable***);**

Description

XmTextFieldSetEditable sets the edit permission state of the TextField widget. When set to True, the text string can be edited.

widget	Specifies the TextField widget ID
editable	Specifies a Boolean value that when True allows text string edits

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3).

XmTextFieldSetHighlight

Purpose

A TextField function that highlights text

Synopsis

#include <Xm/TextF.h>

void XmTextFieldSetHighlight(
    Widget *widget***,**
    XmTextPosition *left***,**
    XmTextPosition *right***,**
    XmHighlightMode *mode***);**

Description

XmTextFieldSetHighlight highlights text between the two specified character positions. The *mode* parameter determines the type of highlighting. Highlighting text merely changes the visual appearance of the text; it does not set the selection.

widget Specifies the TextField widget ID

left Specifies the position of the left boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

right Specifies the position of the right boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

mode Specifies the type of highlighting to be done. A value of **XmHIGHLIGHT_NORMAL** removes highlighting. A value of **XmHIGHLIGHT_SELECTED** highlights the test using reverse video. A value of **XmHIGHLIGHT_SECONDARY_SELECTED** highlights the text using underlining.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3).

XmTextFieldSetInsertionPosition

Purpose

A TextField function that sets the position of the insertion cursor

Synopsis

```
#include <Xm/TextF.h>

void XmTextFieldSetInsertionPosition(
        Widget widget,
        XmTextPosition position);
```

Description

XmTextFieldSetInsertionPosition sets the insertion cursor position of the TextField widget. This routine also calls the widget's **XmNmotionVerifyCallback** callbacks if the insertion cursor position changes.

widget	Specifies the TextField widget ID
position	Specifies the position of the insert cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3).

XmTextFieldSetMaxLength

Purpose

A TextField function that sets the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

```
#include <Xm/TextF.h>

void XmTextFieldSetMaxLength(
        Widget widget,
        int max_length);
```

Description

XmTextFieldSetMaxLength sets the value of the current maximum allowable length of the text string in the TextField widget. The maximum allowable length prevents the user from entering a text string from the keyboard that is larger than this limit. Strings that are entered using the **XmNvalue** (or **XmNvalueWcs**) resource, or the **XmTextFieldSetString** (or **XmTextFieldSetStringWcs**) function ignore this resource.

widget	Specifies the TextField widget ID
max_length	Specifies the maximum allowable length of the text string

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmText(3), **XmTextFieldSetString**(3), and **XmTextFieldSetStringWcs**(3).

XmTextFieldSetSelection

Purpose

A TextField function that sets the primary selection of the text

Synopsis

#include <Xm/TextF.h>

void XmTextFieldSetSelection(
    Widget *widget***,**
    XmTextPosition *first***,**
    XmTextPosition *last***,**
    Time *time***);**

Description

XmTextFieldSetSelection sets the primary selection of the text in the widget. It also sets the insertion cursor position to the last position of the selection and calls the widget's **XmNmotionVerifyCallback** callbacks. **XmTextFieldSetSelection** always generates an **XmNgainPrimaryCallback** unless it fails to take ownership of the primary text selection.

widget	Specifies the TextField widget ID
first	Marks the first character position of the text to be selected
last	Marks the last position of the text to be selected
time	Specifies the time at which the selection value is desired. This should be the same as the time of the event that triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed**.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3).

XmTextFieldSetString

Purpose

A TextField function that sets the string value

Synopsis

#include <Xm/TextF.h>

void XmTextFieldSetString(
    Widget *widget***,**
    char * *value***);**

Description

XmTextFieldSetString sets the string value of the TextField widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. It also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget	Specifies the TextField widget ID
value	Specifies the character pointer to the string value and places the string into the text edit window

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3) and **XmTextFieldSetStringWcs**(3).

XmTextFieldSetStringWcs

Purpose

A TextField function that sets a wide character string value

Synopsis

#include <Xm/TextF.h>

void XmTextFieldSetStringWcs(
    Widget *widget***,**
    wchar_t ***wcstring*);**

Description

XmTextFieldSetStringWcs sets the wide character string value of the TextField widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. It also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget	Specifies the TextField widget ID
wcstring	Specifies the wide character string value

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3) and **XmTextFieldSetString**(3).

XmTextFieldShowPosition

Purpose

A TextField function that forces text at a given position to be displayed

Synopsis

```
#include <Xm/TextF.h>

void XmTextFieldShowPosition(
        Widget widget,
        XmTextPosition position);
```

Description

XmTextFieldShowPosition forces text at the specified position to be displayed. The cursor position is not updated nor is the cursor shown at this position.

widget	Specifies the TextField widget ID
position	Specifies the character position to be displayed. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero). See **XmTextPosition**(3) for details on the **XmTextPosition** data type.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Related Information

XmTextField(3) and **XmTextPosition**(3).

XmTextFieldXYToPos

Purpose

A TextField function that accesses the character position nearest an x and y position

Synopsis

#include <Xm/TextF.h>

XmTextPosition XmTextFieldXYToPos(
  Widget *widget*,
  Position *x*,
  Position *y*);

Description

XmTextFieldXYToPos accesses the character position nearest to the specified *x* and *y* position, relative to the upper left corner of the TextField widget.

widget	Specifies the TextField widget ID
x	Specifies the *x* position, relative to the upper left corner of the widget.
y	Specifies the *y* position, relative to the upper left corner of the widget.

For a complete definition of TextField and its associated resources, see **XmTextField**(3).

Return Values

Returns the character position in the text nearest the *x* and *y* position specified. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

Related Information

XmTextField(3).

XmTextFindString

Purpose

A Text function that finds the beginning position of a text string

Synopsis

#include <Xm/Xm.h>

Boolean XmTextFindString(
    Widget *widget***,**
    XmTextPosition *start***,**
    char ****string*,**
    XmTextDirection *direction***,**
    XmTextPosition ****position*);**

Description

XmTextFindString locates the beginning position of a specified text string. This routine searches forward or backward for the first occurrence of the string starting from the given start position. If it finds a match, the function returns the position of the first character of the string in *position*. If the match string begins at the current position, this routine returns the current position.

widget Specifies the Text widget ID.

start Specifies the character position from which the search proceeds. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

string Specifies the search string.

direction Indicates the search direction. It is relative to the primary direction of the text. The possible values are

XmTEXT_FORWARD
The search proceeds toward the end of the text buffer.

XmTEXT_BACKWARD
The search proceeds toward the beginning of the text buffer.

position Specifies the pointer in which the first character position of the string match is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). If the function returns False, this value is undefined.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns True if a string match is found; otherwise, returns False.

Related Information

XmText(3) and **XmTextFindStringWcs**(3).

XmTextFindStringWcs

Purpose

A Text function that finds the beginning position of a wide character text string

Synopsis

#include <Xm/Text.h>

Boolean XmTextFindStringWcs(
Widget *widget***,**
XmTextPosition *start***,**
wchar_t ****wcstring*,**
XmTextDirection *direction***,**
XmTextPosition ****position*);**

Description

XmTextFindStringWcs locates the beginning position of a specified wide character text string. This routine searches forward or backward for the first occurrence of the string, starting from the given start position. If a match is found, the function returns the position of the first character of the string in *position*. If the match string begins at the current position, this routine returns the current position.

widget	Specifies the Text widget ID.
start	Specifies the character position from which the search proceeds. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
wcstring	Specifies the wide character search string.
direction	Indicates the search direction. It is relative to the primary direction of the text. The possible values are

XmTEXT_FORWARD
The search proceeds toward the end of the buffer.

XmTEXT_BACKWARD
The search proceeds toward the beginning of the buffer.

position Specifies the pointer in which the first character position of the string match is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). If the function returns False, this value is undefined.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns True if a string match is found; otherwise, returns False.

Related Information

XmText(3) and **XmTextFindString**(3).

XmTextGetBaseline

Purpose

A Text function that accesses the y position of the baseline

Synopsis

#include <Xm/Text.h>

int XmTextGetBaseline(
  Widget *widget***);**

Description

XmTextGetBaseline accesses the *y* position of the baseline in the Text widget, relative to the *y* position of the top of the widget.

In vertical mode (when the **XmNlayoutDirection** resource is **XmTOP_TO_BOTTOM**) this function returns 0 and the program should use **XmTextGetCenterline**

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns an integer value that indicates the *y* position of the baseline in the Text widget. The calculation takes into account the margin height, shadow thickness, highlight thickness, and font ascent of the first font (set) in the fontlist used for drawing text. In this calculation, the *y* position of the top of the widget is 0 (zero).

Related Information

XmText(3), **XmTextGetCenterline**(3).

XmTextGetCenterline

Purpose

Return the height (length) of a character string when the writing direction is vertical

Synopsis

#include <Xm/Text.h>

int XmTextGetCenterline(
  Widget *widget***);**

Description

XmTextGetCenterline accesses the x position of the centerline in the **Text** widget, relative to the x position of the top of the widget.

widget Specifies the **Text** widget ID.

Return Values

In the case of horizontal writing, this function accesses 0.

In the case of vertical writing, this function accesses the x position of the first centerline in the Text widget, relative to the x position of the left of the widget. The calculation takes into account the margin width, shadow thickness, highlight thickness, and a half of font width of the first font(set) in the fontlist used for drawing text.

Related Information

XmText(3), **XmTextGetBaseline**(3)

XmTextGetEditable

Purpose

A Text function that accesses the edit permission state

Synopsis

#include <Xm/Text.h>

Boolean XmTextGetEditable(
        Widget *widget***);**

Description

XmTextGetEditable accesses the edit permission state of the Text widget.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns a Boolean value that indicates the state of the **XmNeditable** resource.

Related Information

XmText(3).

XmTextGetInsertionPosition

Purpose

A Text function that accesses the position of the insert cursor

Synopsis

```
#include <Xm/Text.h>

XmTextPosition XmTextGetInsertionPosition(
        Widget widget);
```

Description

XmTextGetInsertionPosition accesses the insertion cursor position of the Text widget.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns an **XmTextPosition** value that indicates the state of the **XmNcursorPosition** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

Related Information

XmText(3).

XmTextGetLastPosition

Purpose

A Text function that accesses the last position in the text

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetLastPosition(
        Widget *widget*);

Description

XmTextGetLastPosition accesses the last position in the text buffer of the Text widget. This is an integer number of characters from the beginning of the buffer, and represents the position that text added to the end of the buffer is placed after. The first character position is 0 (zero). The last character position is equal to the number of characters in the text buffer.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns an **XmTextPosition** value that indicates the last position in the text buffer.

Related Information

XmText(3).

XmTextGetMaxLength

Purpose

A Text function that accesses the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

```
#include <Xm/Text.h>

int XmTextGetMaxLength(
        Widget widget);
```

Description

XmTextGetMaxLength accesses the value of the current maximum allowable length of the text string in the Text widget entered from the keyboard. The maximum allowable length prevents the user from entering a text string larger than this limit. Note that the maximum allowable length is the same as the value of the widget's **XmNmaxLength** resource.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns the integer value that indicates the string's maximum allowable length that can be entered from the keyboard.

Related Information

XmText(3).

XmTextGetSelection

Purpose

A Text function that retrieves the value of the primary selection

Synopsis

```
#include <Xm/Text.h>

char * XmTextGetSelection(
        Widget widget);
```

Description

XmTextGetSelection retrieves the value of the primary selection. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns a character pointer to the string that is associated with the primary selection.

Related Information

XmText(3) and **XmTextGetSelectionWcs**(3).

XmTextGetSelectionPosition

Purpose

A Text function that accesses the position of the primary selection

Synopsis

#include <Xm/Text.h>

Boolean XmTextGetSelectionPosition(
    Widget *widget***,**
    XmTextPosition **left***,**
    XmTextPosition **right***);**

Description

XmTextGetSelectionPosition accesses the left and right position of the primary selection in the text buffer of the Text widget.

widget	Specifies the Text widget ID
left	Specifies the pointer in which the position of the left boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).
right	Specifies the pointer in which the position of the right boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns True if the widget owns the primary selection; otherwise, it returns False.

Related Information

XmText(3).

XmTextGetSelectionWcs

Purpose

A Text function that retrieves the value of a wide character encoded primary selection

Synopsis

#include <Xm/Text.h>

wchar_t * XmTextGetSelectionWcs(
Widget *widget***);**

Description

XmTextGetSelectionWcs retrieves the value of the primary selection that is encoded in a wide character format. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the wide character buffer by calling **XtFree**.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns the wide character string that is associated with the primary selection in the Text widget.

Related Information

XmText(3) and **XmTextGetSelection**(3).

XmTextGetSource

Purpose

A Text function that accesses the source of the widget

Synopsis

#include <Xm/Text.h>

XmTextSource XmTextGetSource(
  Widget *widget***);**

Description

XmTextGetSource accesses the source of the Text widget. Text widgets can share sources of text so that editing in one widget is reflected in another. This function accesses the source of one widget so that it can be made the source of another widget, using the function **XmTextSetSource**(3).

Setting a new text source destroys the old text source if no other Text widgets are using that source. To replace a text source but keep it for later use, create an unmanaged Text widget and set its source to the text source you want to keep.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns an **XmTextSource** value that represents the source of the Text widget.

Related Information

XmText(3).

XmTextGetString

Purpose

A Text function that accesses the string value

Synopsis

#include <Xm/Text.h>

char * XmTextGetString(
    Widget *widget***);**

Description

XmTextGetString accesses the string value of the Text widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns a character pointer to the string value of the text widget. This returned value is a copy of the value of the **XmNvalue** resource. Returns an empty string if the length of the Text widget's string is 0 (zero).

Related Information

XmText(3) and **XmTextGetStringWcs**(3).

XmTextGetStringWcs

Purpose

A Text function that retrieves a copy of the wide character string value of a Text widget

Synopsis

#include <Xm/Text.h>

wchar_t * XmTextGetStringWcs(
Widget *widget***);**

Description

XmTextGetStringWcs retrieves a copy of the wide character string value of the Text widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget	Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns the wide character string value of the Text widget. The function returns an empty string if the length of the Text widget's string is 0 (zero).

Related Information

XmText(3) and **XmTextGetString**(3).

XmTextGetSubstring

Purpose

A Text function that retrieves a copy of a portion of the internal text buffer

Synopsis

#include <Xm/Text.h>

int XmTextGetSubstring(
 Widget *widget***,**
 XmTextPosition *start***,**
 int *num_chars***,**
 int *buffer_size***,**
 char ****buffer*);**

Description

XmTextGetSubstring retrieves a copy of a portion of the internal text buffer of a Text widget. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

The size of the required buffer depends on the maximum number of bytes per character (**MB_CUR_MAX**) for the current locale. **MB_CUR_MAX** is a macro defined in **stdlib.h**. The buffer should be large enough to contain the substring to be copied and a NULL terminator. Use the following equation to calculate the size of buffer the application should provide:
buffer_size **= (***num_chars**** MB_CUR_MAX) + 1**

widget	Specifies the Text widget ID.
start	Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
num_chars	Specifies the number of characters to be copied into the provided buffer.
buffer_size	Specifies the size of the supplied buffer in bytes. This size should account for a NULL terminator.

buffer Specifies the character buffer into which the internal text buffer will be copied.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget’s buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmText(3) and **XmTextGetSubstringWcs**(3).

XmTextGetSubstringWcs

Purpose

A Text function that retrieves a portion of a wide character internal text buffer

Synopsis

#include <Xm/Text.h>

int XmTextGetSubstringWcs(
  Widget *widget***,**
  XmTextPosition *start***,**
  int *num_chars***,**
  int *buffer_size***,**
  wchar_t ****buffer*);**

Description

XmTextGetSubstringWcs retrieves a copy of a portion of the internal text buffer of a Text widget that is stored in a wide character format. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

widget	Specifies the Text widget ID.
start	Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
num_chars	Specifies the number of **wchar_t** characters to be copied into the provided buffer.
buffer_size	Specifies the size of the supplied buffer as a number of **wchar_t** storage locations. The minimum size is *num_chars* + 1.
buffer	Specifies the wide character buffer into which the internal text buffer will be copied.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmText(3) and **XmTextGetSubstring**(3).

XmTextGetTopCharacter

Purpose

A Text function that accesses the position of the first character displayed

Synopsis

```
#include <Xm/Text.h>

XmTextPosition XmTextGetTopCharacter(
        Widget widget);
```

Description

XmTextGetTopCharacter accesses the position of the text at the top of the Text widget. If there is no text in the Text widget (in other words, **XmNvalue** contains an empty string), then **XmTextGetTopCharacter** returns 0.

Suppose that the **XmNtopCharacter** resource has been set to a value greater than the number of characters in the text widget. In this case, **XmTextGetTopCharacter** returns an **XmTextPosition** value identifying the position of the first character in the last line of text in a multiline case; otherwise, it identifies the position of the last character in the line.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns an **XmTextPosition** value that indicates the state of the **XmNtopCharacter** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

Related Information

XmText(3).

XmTextInsert

Purpose

A Text function that inserts a character string into a text string

Synopsis

#include <Xm/Text.h>

void XmTextInsert(
    Widget *widget***,**
    XmTextPosition *position***,**
    char * *value***);**

Description

XmTextInsert inserts a character string into the text string in the Text widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the parameter *position* must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. If the **XmNcursorPosition** resource is greater than or is the same value as *position*, the **XmNmotionVerifyCallback** is called.

Note that, if *value* is a null string, no callbacks will be generated, since no modifications will have been made.

widget	Specifies the Text widget ID.
position	Specifies the position in the text string where the character string is to be inserted.
value	Specifies the character string value to be added to the text widget.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3) and **XmTextInsertWcs**(3).

XmTextInsertWcs

Purpose

A Text function that inserts a wide character string into a Text widget

Synopsis

#include <Xm/Text.h>

void XmTextInsertWcs(
  Widget *widget***,**
  XmTextPosition *position***,**
  wchar_t ****wcstring*);**

Description

XmTextInsertWcs inserts a wide character string into the Text widget at a specified location. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. If the **XmNcursorPosition** resource is greater than or is the same value as *position*, the **XmNmotionVerifyCallback** is called.

Note that, if *value* is a null string, no callbacks will be generated, since no modifications will have been made.

widget	Specifies the Text widget ID
position	Specifies the position in the text string where the new character string is to be inserted
wcstring	Specifies the wide character string value to be added to the Text widget

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3) and **XmTextInsert**(3).

XmTextPaste

Purpose

A Text function that inserts the clipboard selection

Synopsis

#include <Xm/Text.h>

Boolean XmTextPaste(
Widget *widget***);**

Description

XmTextPaste inserts the clipboard selection at the insertion cursor of the destination widget. If **XmNpendingDelete** is True and the insertion cursor is inside the current selection, the clipboard selection replaces the selected text.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. If the **XmNcursorPosition** resource is greater than or is the same value as the position where the selection is to be inserted, the **XmNmotionVerifyCallback** is called.

This routine calls the widget's **XmNdestinationCallback** procedures with the *selection* member of the **XmDestinationCallbackStruct** set to *CLIPBOARD* and with the *operation* member set to **XmCOPY**.

widget	Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns False if no transfers take place. Otherwise, it returns True.

Related Information

XmText(3).

XmTextPasteLink

Purpose

A Text function that inserts a link to the clipboard selection

Synopsis

#include <Xm/Text.h>

Boolean XmTextPasteLink(
    Widget *widget***);**

Description

XmTextPasteLink inserts a link to the clipboard selection at the insertion cursor. This routine calls the widget's **XmNdestinationCallback** procedures with the *selection* member of the **XmDestinationCallbackStruct** set to *CLIPBOARD* and with the *operation* member set to **XmLINK**. The Text widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the clipboard selection and for taking any related actions.

widget	Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns False if no transfers take place. Otherwise, it returns True.

Related Information

XmText(3).

XmTextPosToXY

Purpose

A Text function that accesses the x and y position of a character position

Synopsis

#include <Xm/Text.h>

Boolean XmTextPosToXY(
    Widget *widget*,
    XmTextPosition *position*,
    Position **x*,
    Position **y*);

Description

XmTextPosToXY accesses the *x* and *y* position, relative to the upper left corner of the Text widget, of a given character position in the text buffer.

In the case of horizontal writing, the position is the origin of the character. In the case of vertical writing, the position is the vertical origin of the character.

widget	Specifies the Text widget ID
position	Specifies the character position in the text for which the *x* and *y* position is accessed. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).
x	Specifies the pointer in which the *x* position is returned. The returned position is the distance from the left side of the widget to the left border of the character. This value is meaningful only if the function returns True.
y	Specifies the pointer in which the *y* position is returned. The returned position is the distance from the top of the widget to the character's baseline. This value is meaningful only if the function returns True.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns True if the character position is displayed in the Text widget; otherwise, it returns False, and no *x* or *y* value is returned.

Related Information

XmText(3).

XmTextRemove

Purpose

A Text function that deletes the primary selection

Synopsis

```
#include <Xm/Text.h>

Boolean XmTextRemove(
          Widget widget);
```

Description

XmTextRemove deletes the primary selected text. If there is a selection, this routine also calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This function may also call the **XmNmotionVerifyCallback** callback.

widget	Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

This function returns False if the primary selection is NULL or if the *widget* does not own the primary selection. Otherwise, it returns True.

Related Information

XmText(3).

XmTextReplace

Purpose

A Text function that replaces part of a text string

Synopsis

#include <Xm/Text.h>

void XmTextReplace(
    Widget *widget***,**
    XmTextPosition *from_pos***,**
    XmTextPosition *to_pos***,**
    char * *value***);**

Description

XmTextReplace replaces part of the text string in the Text widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both parameters, *from_pos* and *to_pos*, must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. The **XmNmotionVerifyCallback** is generated if *to_pos* is less than or equal to the cursor position and the length of *value* is not the same as the length of the text being replaced, or if the cursor position is between *from_pos* and *to_pos*, and the distance from the cursor position to *from_pos* is greater than the length of *value*.

widget	Specifies the Text widget ID
from_pos	Specifies the start position of the text to be replaced

to_pos	Specifies the end position of the text to be replaced
value	Specifies the character string value to be added to the text widget

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3) and **XmTextReplaceWcs**(3).

XmTextReplaceWcs

Purpose

A Text function that replaces part of a wide character string in a Text widget

Synopsis

#include <Xm/Text.h>

void XmTextReplaceWcs(
  Widget *widget***,**
  XmTextPosition *from_pos***,**
  XmTextPosition *to_pos***,**
  wchar_t ***wcstring);**

Description

XmTextReplaceWcs replaces part of the wide character string in the Text widget. The character positions begin at zero and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the *from_pos* parameter must be 1 and the *to_pos* parameter must be 3. To insert a string after the fourth character, both the *from_pos* and *to_pos* parameters must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This routine calls the widget's **XmNmotionVerifyCallback** callback when *from_pos* is less than or equal to the cursor position.

widget	Specifies the Text widget ID
from_pos	Specifies the start position of the text to be replaced
to_pos	Specifies the end position of the text to be replaced
wcstring	Specifies the wide character string value to be added to the Text widget

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3) and **XmTextReplace**(3).

XmTextScroll

Purpose

A Text function that scrolls text

Synopsis

#include <Xm/Text.h>

void XmTextScroll(
    Widget *widget***,**
    int *lines***);**

Description

XmTextScroll scrolls text by a given number of lines in a Text widget. The sign of the number is interpreted according to the value of the *XmNlayoutDirection* resource.

widget Specifies the Text widget ID

lines Specifies the number of lines of text to scroll. A positive value causes text to scroll upward; a negative value causes text to scroll downward. Note that the text will scroll only as long as one line of text remains visible in the window.

If a navigator exists, this function uses the *XmQTnavigator* trait to update the vertical navigator's value.

In the case of vertical writing, a positive value causes the text to scroll forward; a negative value causes the lines to scroll backward.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetAddMode

Purpose

A Text function that sets the state of Add mode

Synopsis

#include <Xm/Text.h>

void XmTextSetAddMode(
Widget *widget***,**
Boolean *state***);**

Description

XmTextSetAddMode controls whether or not the Text widget is in Add mode. When the widget is in Add mode, the insert cursor can be moved without disturbing the primary selection.

widget	Specifies the Text widget ID
state	Specifies whether or not the widget is in Add mode. A value of True turns on Add mode; a value of False turns off Add mode.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetEditable

Purpose

A Text function that sets the edit permission

Synopsis

```
#include <Xm/Text.h>

void XmTextSetEditable(
        Widget widget,
        Boolean editable);
```

Description

XmTextSetEditable sets the edit permission state of the Text widget. When set to True, the text string can be edited.

widget	Specifies the Text widget ID
editable	Specifies a Boolean value that when True allows text string edits

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetHighlight

Purpose

A Text function that highlights text

Synopsis

```
#include <Xm/Text.h>

void XmTextSetHighlight(
        Widget widget,
        XmTextPosition left,
        XmTextPosition right,
        XmHighlightMode mode);
```

Description

XmTextSetHighlight highlights text between the two specified character positions. The *mode* parameter determines the type of highlighting. Highlighting text merely changes the visual appearance of the text; it does not set the selection.

widget	Specifies the Text widget ID
left	Specifies the position of the left boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
right	Specifies the position of the right boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).
mode	Specifies the type of highlighting to be done. A value of **XmHIGHLIGHT_NORMAL** removes highlighting. A value of **XmHIGHLIGHT_SELECTED** highlights the text using reverse video. A value of **XmHIGHLIGHT_SECONDARY_SELECTED** highlights the text using underlining.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetInsertionPosition

Purpose

A Text function that sets the position of the insert cursor

Synopsis

```
#include <Xm/Text.h>

void XmTextSetInsertionPosition(
        Widget widget,
        XmTextPosition position);
```

Description

XmTextSetInsertionPosition sets the insertion cursor position of the Text widget. This routine also calls the widget's **XmNmotionVerifyCallback** callbacks if the insertion cursor position changes.

widget	Specifies the Text widget ID
position	Specifies the position of the insertion cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetMaxLength

Purpose

A Text function that sets the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

#include <Xm/Text.h>

void XmTextSetMaxLength(
  Widget *widget***,**
  int *max_length***);**

Description

XmTextSetMaxLength sets the value of the current maximum allowable length of the text string in the Text widget. The maximum allowable length prevents the user from entering a text string from the keyboard that is larger than this limit. Strings that are entered using the **XmNvalue** (or **XmNvalueWcs**) resource, or the **XmTextSetString** (or **XmTextSetStringWcs**) function ignore this resource.

widget	Specifies the Text widget ID
max_length	Specifies the maximum allowable length of the text string

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3), **XmTextSetString**(3), and **XmTextSetStringWcs**(3).

XmTextSetSelection

Purpose

A Text function that sets the primary selection of the text

Synopsis

#include <Xm/Text.h>

void XmTextSetSelection(
  Widget *widget***,**
  XmTextPosition *first***,**
  XmTextPosition *last***,**
  Time *time***);**

Description

XmTextSetSelection sets the primary selection of the text in the widget. It also sets the insertion cursor position to the last position of the selection and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget	Specifies the Text widget ID
first	Marks the first character position of the text to be selected
last	Marks the last position of the text to be selected
time	Specifies the time at which the selection value is desired. This should be the same as the time of the event that triggered this request. request. One source of a valid time stamp is the function **XtLastTimestampProcessed**.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetSource

Purpose

A Text function that sets the source of the widget

Synopsis

#include <Xm/Text.h>

void XmTextSetSource(
Widget *widget***,**
XmTextSource *source***,**
XmTextPosition *top_character***,**
XmTextPosition *cursor_position***);**

Description

XmTextSetSource sets the source of the Text widget. Text widgets can share sources of text so that editing in one widget is reflected in another. This function sets the source of one widget so that it can share the source of another widget.

Setting a new text source destroys the old text source if no other Text widgets are using that source. To replace a text source but keep it for later use, create an unmanaged Text widget and set its source to the text source you want to keep.

widget Specifies the Text widget ID.

source Specifies the source with which the widget displays text. This can be a value returned by the **XmTextGetSource**(3) function. If no source is specified, the widget creates a default string source.

top_character
Specifies the position in the text to display at the top of the widget. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

cursor_position
Specifies the position in the text at which the insert cursor is located. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextSetString

Purpose

A Text function that sets the string value

Synopsis

```
#include <Xm/Text.h>

void XmTextSetString(
        Widget widget,
        char * value);
```

Description

XmTextSetString sets the string value of the Text widget. This routine calls the widget’s **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This function also sets the insertion cursor position to the beginning of the string and calls the widget’s **XmNmotionVerifyCallback** callbacks.

widget	Specifies the Text widget ID
value	Specifies the character pointer to the string value and places the string into the text edit window

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3) and **XmTextSetStringWcs**(3).

XmTextSetStringWcs

Purpose

A Text function that sets a wide character string value

Synopsis

#include <Xm/Text.h>

void XmTextSetStringWcs(
        Widget *widget***,**
        wchar_t ****wcstring*);**

Description

XmTextSetStringWcs sets the wide character string value of the Text widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This function also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget	Specifies the Text widget ID
value	Specifies the wide character string value

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3) and **XmTextSetString**(3).

XmTextSetTopCharacter

Purpose

A Text function that sets the position of the first character displayed

Synopsis

#include <Xm/Text.h>

void XmTextSetTopCharacter(
    Widget *widget***,**
    XmTextPosition *top_character***);**

Description

XmTextSetTopCharacter sets the position of the text at the top of the Text widget. If the **XmNeditMode** is **XmMULTI_LINE_EDIT**, the line of text that contains *top_character* is displayed at the top of the widget without the text shifting left or right. If the edit mode is **XmSINGLE_LINE_EDIT**, the text moves horizontally so that *top_character* is the first character displayed.

widget Specifies the Text widget ID

top_character
Specifies the position in the text to display at the top of the widget. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextShowPosition

Purpose

A Text function that forces text at a given position to be displayed

Synopsis

```
#include <Xm/Text.h>

void XmTextShowPosition(
        Widget widget,
        XmTextPosition position);
```

Description

XmTextShowPosition forces text at the specified position to be displayed. If the **XmNautoShowCursorPosition** resource is True, the application should also set the insert cursor to this position.

widget Specifies the Text widget ID

position Specifies the character position to be displayed. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

If a navigator exists, this function uses the *XmQTnavigator* trait to update the horizontal navigator's value.

For a complete definition of Text and its associated resources, see **XmText**(3).

Related Information

XmText(3).

XmTextXYToPos

Purpose

A Text function that accesses the character position nearest an x and y position

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextXYToPos(
Widget *widget*,
Position *x*,
Position *y*);

Description

XmTextXYToPos accesses the character position nearest to the specified *x* and *y* position, relative to the upper left corner of the Text widget.

In the case of horizontal writing, the position is the origin of the character. In the case of vertical writing, the position is the vertical origin of the character.

widget	Specifies the Text widget ID
x	Specifies the *x* position, relative to the upper left corner of the widget
y	Specifies the *y* position, relative to the upper left corner of the widget

For a complete definition of Text and its associated resources, see **XmText**(3).

Return Values

Returns the character position in the text nearest the *x* and *y* position specified. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

Related Information

XmText(3).

XmToggleButtonGadgetGetState

Purpose

A ToggleButtonGadget function that obtains the state of a ToggleButtonGadget

Synopsis

#include <Xm/ToggleBG.h>

Boolean XmToggleButtonGadgetGetState(
Widget *widget***);**

Description

XmToggleButtonGadgetGetState obtains the state of a ToggleButtonGadget.

widget	Specifies the ToggleButtonGadget ID

For a complete definition of ToggleButtonGadget and its associated resources, see **XmToggleButtonGadget**(3).

Return Values

Returns True if the button is selected and False if the button is unselected.

Related Information

XmToggleButtonGadget(3).

XmToggleButtonGadgetSetState

Purpose

A ToggleButtonGadget function that sets or changes the current state

Synopsis

#include <Xm/ToggleBG.h>

void XmToggleButtonGadgetSetState(
    Widget *widget***,**
    Boolean *state***,**
    Boolean *notify***);**

Description

XmToggleButtonGadgetSetState sets or changes the ToggleButtonGadget's current state.

widget	Specifies the ToggleButtonGadget widget ID.
state	Specifies a Boolean value that indicates whether the ToggleButtonGadget state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected.
notify	Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButtonGadget. When this argument is True and the ToggleButtonGadget is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButtonGadget causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButtonGadget and its associated resources, see **XmToggleButtonGadget**(3).

Related Information

XmToggleButtonGadget(3).

XmToggleButtonGetState

Purpose

A ToggleButton function that obtains the state of a ToggleButton

Synopsis

#include <Xm/ToggleB.h>

```
Boolean XmToggleButtonGetState(
        Widget widget);
```

Description

XmToggleButtonGetState obtains the state of a ToggleButton.

widget Specifies the ToggleButton widget ID

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton**(3).

Return Values

Returns True if the button is selected and False if the button is unselected.

Related Information

XmToggleButton(3).

XmToggleButtonSetState

Purpose

A ToggleButton function that sets or changes the current state

Synopsis

#include <Xm/ToggleB.h>

void XmToggleButtonSetState(
    Widget *widget***,**
    Boolean *state***,**
    Boolean *notify***);**

Description

XmToggleButtonSetState sets or changes the ToggleButton's current state.

widget	Specifies the ToggleButton widget ID.
state	Specifies a Boolean value that indicates whether the ToggleButton state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected.
notify	Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButton. When this argument is True and the ToggleButton is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButton causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton**(3).

Related Information

XmToggleButton(3).

XmToggleButtonSetValue

Purpose

A ToggleButton function that sets or changes the current state

Synopsis

#include <Xm/ToggleB.h>

```
void XmToggleButtonSetValue(
        Widget widget,
        XmToggleButtonState state,
        Boolean notify);
```

Description

XmToggleButtonSetValue sets or changes the ToggleButton's current state.

widget	Specifies the ToggleButton widget ID.
state	Specifies whether the ToggleButton state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected, if it is **XmINDETERMINATE**, the button state is neither.
notify	Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButton. When this argument is True and the ToggleButton is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButton causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton**(3).

Related Information

XmToggleButton(3).

XmTrackingEvent

Purpose

A Toolkit function that provides a modal interaction

Synopsis

#include <Xm/Xm.h>

Widget XmTrackingEvent(
    Widget *widget*,
    Cursor *cursor*,
    Boolean *confine_to*,
    XEvent ***event_return*);

Description

XmTrackingEvent provides a modal interface for selection of a component. It is intended to support context help. The function calls the **XmUpdateDisplay** function. **XmTrackingEvent** then grabs the pointer and discards succeeding events until **BSelect** is released or a key is pressed and then released. The function then returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released, and ungrabs the pointer.

widget	Specifies the widget ID of a widget to use as the basis of the modal interaction. That is, the widget within which the interaction must occur, usually a top-level shell.
cursor	Specifies the cursor to be used for the pointer during the interaction. This is a standard X cursor name.
confine_to	Specifies whether or not the cursor should be confined to *widget*.
event_return	Returns the ButtonRelease or KeyRelease event that causes the function to return.

Return Values

Returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released. If no widget or gadget contains the pointer, the function returns NULL.

Related Information

XmTrackingLocate(3).

XmTrackingLocate

Purpose

A Toolkit function that provides a modal interaction

Synopsis

#include <Xm/Xm.h>

Widget XmTrackingLocate(
  Widget *widget***,**
  Cursor *cursor***,**
  Boolean *confine_to***);**

Description

XmTrackingLocate provides a modal interface for selection of a component. It is intended to support context help. This function is implemented as **XmTrackingEvent**.

NOTE: This function is obsolete and exists for compatibility with previous releases. It has been replaced by **XmTrackingEvent**.

widget	Specifies the widget ID of a widget to use as the basis of the modal interaction. That is, the widget within which the interaction must occur, usually a top-level shell.
cursor	Specifies the cursor to be used for the pointer during the interaction. This is a standard X cursor name.
confine_to	Specifies whether or not the cursor should be confined to *widget*

Return Values

Returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released. If no widget or gadget contains the pointer, the function returns NULL.

Related Information

XmTrackingEvent(3).

XmTransferDone

Purpose

A toolkit function that completes a data transfer

Synopsis

#include <Xm/Xm.h>

void XmTransferDone(
    XtPointer *transfer_id***,**
    XmTransferStatus *status***);**

Description

XmTransferDone completes an already-initiated data transfer operation. An application can call this routine from an **XmNdestinationCallback** procedure or any function called as a result, including the selection procedures called as a result of calls to **XmTransferValue**.

The caller of **XmTransferDone** supplies an identifier for the transfer operation and an indication of the completion status. **XmTransferDone** causes any remaining transfers for the operation to be discarded.

transfer_id Specifies a unique indentifier for the data transfer operation. The value must be the same as the value of the **transfer_id** member of the **XmDestinationCallbackStruct** passed to the **XmNdestinationCallback** procedure.

status Specifies the completion status of the data transfer. Following are the possible values:

XmTRANSFER_DONE_SUCCEED
The transfer was completed successfully. This status has the following additional effects:

- For a move operation, the selection owner receives a request to convert the selection to the *DELETE* target.

- If a *TRANSACT* operation is in progress, the owner receives a request to commit the transaction.
- If a *PERSIST* or _MOTIF_SNAPSHOT operation is in progress, the owner receives a notification that the operation is finished.
- The widget class destination procedure is not called.

XmTRANSFER_DONE_FAIL
The transfer was completed unsuccessfully. This status has the following additional effects:

- For a move operation, the selection owner does not receive a request to convert the selection to the *DELETE* target.
- For a drag and drop operation, the DropTransfer's **XmNtransferStatus** is set to **XmTRANSFER_FAILURE**.
- If a *TRANSACT* operation is in progress, the owner receives a request to abort the transaction.
- If a *PERSIST* or _MOTIF_SNAPSHOT operation is in progress, the owner receives a notification that the operation is finished.
- The widget class destination procedure is not called.

XmTRANSFER_DONE_CONTINUE
This status has the same effect as **XmTRANSFER_DONE_SUCCEED**, except that if a *PERSIST* or _MOTIF_SNAPSHOT operation is in progress, the owner does not receive a notification that the operation is finished.

XmTRANSFER_DONE_DEFAULT
The widget class destination procedure is called. Further effects depend on the actions of that procedure.

Related Information

XmTransferSendRequest(3), **XmTransferStartRequest**(3), **XmTransferStartRequest**(3), and **XmTransferValue**(3).

XmTransferSendRequest

Purpose

A toolkit function that transfers a MULTIPLE request

Synopsis

#include <Xm/Transfer.h>

void XmTransferSendRequest(
    XtPointer *transfer_id***,**
    Time *time***);**

Description

XmTransferSendRequest marks the end of a MULTIPLE request started by **XmTransferStartRequest**.

transfer_id Specifies a unique indentifier for the data transfer operation.

time Specifies the time of the *XEvent* that triggered the data transfer. You should typically set this field to *XtLastTimestampProcessed*.

Related Information

XmTransferSetParameters(3), **XmTransferStartRequest**(3), and **XmTransferValue**(3).

XmTransferSetParameters

Purpose

A toolkit function that establishes parameters to be passed by the next call to XmTransferValue

Synopsis

#include <Xm/Transfer.h>

void XmTransferSetParameters(
XtPointer *transfer_id***,**
XtPointer *parm***,**
int *parm_fmt***,**
unsigned long *parm_length***,**
Atom *parm_type***);**

Description

XmTransferSetParameters establishes a parameter definition. Your application calls **XmTransferSetParameters** just before calling **XmTransferValue**, and only if **XmTransferValue** needs to transfer a value containing a parameter.

transfer_id Specifies a unique indentifier for the data transfer operation. The value must be the same as the value of the **transfer_id** member of the **XmDestinationCallbackStruct** passed to the **XmNdestinationCallback** procedure.

parm Specifies parameters to be passed to the conversion routine (and the **XmNconvertCallback** procedures, if any) of the widget that owns the selection. The type and length of parameters are target-specific. If the target takes no parameters, the value is NULL.

parm_fmt Specifies whether the data in *parm* should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities. Possible values are 0 (when *parm* is NULL), 8, 16, and 32.

parm_length Specifies the number of elements of data in *parm*, where each element has the number of bits specified by *parm_fmt*. When *parm* is NULL, the value is 0.

parm_type Specifies the type of *parm*.

Related Information

XmTransferSendRequest(3), **XmTransferStartRequest**(3), and **XmTransferValue**(3).

XmTransferStartRequest

Purpose

A toolkit function that begins a MULTIPLE transfer

Synopsis

#include <Xm/Transfer.h>

void XmTransferStartRequest(
    XtPointer *transfer_id***);**

Description

XmTransferStartRequest begins a *MULTIPLE* request. The *MULTIPLE* request may contain one or more calls to **XmTransferValue**. Your application concludes a *MULTIPLE* request by calling **XmTransferSendRequest**.

XmTransferStartRequest is typically called by a destination callback or by a transfer procedure.

transfer_id Specifies a unique indentifier for the data transfer operation. You should use the **transfer_id** passed in the **XmDestinationCallbackStruct** or **XmSelectionCallbackStruct**.

Related Information

XmTransferSetParameters(3), **XmTransferSendRequest**(3), and **XmTransferValue**(3).

XmTransferValue

Purpose

A toolkit function that transfers data to a destination

Synopsis

#include <Xm/Xm.h>

void XmTransferValue(
    XtPointer *transfer_id***,**
    Atom *target***,**
    XtCallbackProc *proc***,**
    XtPointer *client_data***,**
    Time *time***);**

Description

XmTransferValue converts a selection, transferring any data from the selection owner, in the context of an already-initiated data transfer operation. An application can call this routine from an **XmNdestinationCallback** procedure or any function called as a result.

The caller of **XmTransferValue** supplies the target to which the selection is converted. The caller also supplies a callback procedure to handle the data that results from the conversion.

transfer_id Specifies a unique indentifier for the data transfer operation. The value must be the same as the value of the **transfer_id** member of the **XmDestinationCallbackStruct** passed to the **XmNdestinationCallback** procedure.

target Specifies the target to which the selection is to be converted.

proc Specifies a callback procedure to be invoked when the selection has been converted and the data, if any, is available. This procedure is responsible for inserting or otherwise handling any data transferred. The procedure can also terminate the data transfer by calling **XmTransferDone**. The *proc* receives three arguments:

- The widget that requested the conversion
- The value of the *client_data* argument
- A pointer to an **XmSelectionCallbackStruct**

This procedure can be called before or after **XmTransferValue** returns.

client_data Specifies data to be passed to the callback procedure (the value of the *proc* argument) when the selection has been converted.

time Specifies the time of the *XEvent* that triggered the data transfer. You should typically set this field to **XtLastTimestampProcessed**.

The callback procedure (the value of the *proc* argument) receives a pointer to an **XmSelectionCallbackStruct**, which has the following definition:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        Atom target;
        Atom type;
        XtPointer transfer_id;
        int flags;
        int remaining;
        XtPointer value;
        unsigned long length;
        int format;
} XmSelectionCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Specifies the selection that has been converted.

target Specifies the target to which **XmTransferValue** requested conversion. The value is the same as the value of the *target* argument to **XmTransferValue**.

type Specifies the type of the selection value. This is not the target, but the type used to represent the target. The value *XT_CONVERT_FAIL* means that the selection owner did not respond to the conversion request within the Intrinsics selection timeout interval.

transfer_id	Specifies a unique indentifier for the data transfer operation. The value is the same as the value of the **transfer_id** argument to **XmTransferValue**.
flags	This member is currently unused. The value is always **XmSELECTION_DEFAULT**.
remaining	Indicates the number of transfers remaining for the operation specified by **transfer_id**.
value	Represents the data transferred by this request. The application is responsible for freeing the value by calling **XtFree**.
length	Indicates the number of elements of data in *value*, where each element has the size symbolized by *format*. If *value* is NULL, *length* is 0.
format	Indicates whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).

Related Information

XmTransferSetParameters(3), **XmTransferSendRequest**(3), and **XmTransferStartRequest**(3).

XmTranslateKey

Purpose

The default keycode-to-keysym translator

Synopsis

#include <Xm/Xm.h>

void XmTranslateKey(
  **Display ****display*,
  KeyCode *keycode*,
  Modifiers *modifiers*,
  **Modifiers ****modifiers_return*,
  KeySym ****keysym_return*);**

Description

XmTranslateKey is the default *XtKeyProc* translation procedure for Motif applications. The function takes a keycode and modifiers and returns the corresponding keysym.

XmTranslateKey serves two main purposes: to enable new translators with expanded functionality to get the default Motif keycode-to-keysym translation in addition to whatever they add, and to reinstall the default translator. This function enables keysyms defined by the Motif virtual bindings to be used when an application requires its own XtKeyProc to be installed.

display	Specifies the display that the keycode is from
keycode	Specifies the keycode to translate
modifiers	Specifies the modifier keys to be applied to the keycode
modifiers_return	Specifies a mask of the modifier keys actually used to generate the keysym (an AND of *modifiers* and any default modifiers applied by the currently registered translator)
keysym_return	Specifies a pointer to the resulting keysym

Related Information

VirtualBindings(3).

XmUninstallImage

Purpose

A pixmap caching function that removes an image from the image cache

Synopsis

#include <Xm/Xm.h>

Boolean XmUninstallImage(
    XImage * *image***);**

Description

XmUninstallImage removes an image from the image cache.

image	Points to the image structure given to the XmInstallImage() routine

Return Values

Returns True when successful; returns False if the *image* is NULL, or if it cannot be found to be uninstalled.

Related Information

XmInstallImage(3), **XmGetPixmap**(3), and **XmDestroyPixmap**(3).

XmUpdateDisplay

Purpose

A function that processes all pending exposure events immediately

Synopsis

void XmUpdateDisplay (*widget***)**
    Widget *widget***;**

Description

XmUpdateDisplay provides the application with a mechanism for forcing all pending exposure events to be removed from the input queue and processed immediately. When a user selects a button within a menu pane, the menu panes are unposted and then any activation callbacks registered by the application are invoked. If one of the callbacks performs a time-consuming action, the portion of the application window that was covered by the menu panes will not have been redrawn; normal exposure processing does not occur until all of the callbacks have been invoked. If the application writer suspects that a callback will take a long time, then the callback may choose to invoke **XmUpdateDisplay** before starting its time-consuming operation. This function is also useful any time a transient window, such as a dialog box, is unposted; callbacks are invoked before normal exposure processing can occur.

widget	Specifies any widget or gadget.

XmVaCreateSimpleCheckBox

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmVaCreateSimpleCheckBox(
  Widget *parent***,**
  String *name***,**
  XtCallbackProc *callback***);**

Description

XmVaCreateSimpleCheckBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (*varargs*) calling convention.

This routine creates a CheckBox and its ToggleButtonGadget children. A CheckBox is similar to a RadioBox, except that more than one button can be selected at a time. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. Buttons are named and created in the order in which they are specified in the variable portion of the argument list.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
callback	Specifies a callback procedure to be called when a button's value changes. This callback function is added to each button after creation as the button's **XmNvalueChangedCallback**. The callback function is called when a button's value changes, and the button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. The following list describes the possible first arguments in each group of *varargs*:

XmVaCHECKBUTTON
This is followed by four additional arguments. The set specifies one button in the CheckBox and some of its resource values. The following list describes the additional four arguments, in order.

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**. This is ignored in this release.
accelerator	The accelerator, of type **String**. This is ignored in this release.
accelerator_text	The accelerator text, of type **XmString**. This is ignored in this release.

resource_name
This is followed by one additional argument, the value of the resource, of type *XtArgVal*. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type *XtArgVal*
size	The size of the resource value in bytes, of type *int*

XtVaNestedList
This is followed by one additional argument of type XtVarArgsList. This argument is a nested list of *varargs* returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3), **XmCreateRowColumn**(3), **XmCreateSimpleCheckBox**(3), **XmCreateSimpleRadioBox**(3), **XmRowColumn**(3), and **XmVaCreateSimpleRadioBox**(3).

XmVaCreateSimpleMenuBar

Purpose

A RowColumn widget convenience creation function

Synopsis

```
#include <Xm/RowColumn.h>

Widget XmVaCreateSimpleMenuBar(
        Widget parent,
        String name);
```

Description

XmVaCreateSimpleMenuBar creates an instance of a RowColumn widget of type **XmMENU_BAR** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (*varargs*) calling convention.

This routine creates a MenuBar and its CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. Buttons are named and created in the order in which they are specified in the variable portion of the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of *varargs*:

XmVaCASCADEBUTTON
This is followed by two additional arguments. The set specifies one button in the MenuBar and some of its resource values. Following are the additional two arguments, in order:

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**

resource_name
: This is followed by one additional argument, the value of the resource, of type *XtArgVal*. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
: This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type *XtArgVal*
size	The size of the resource value in bytes, of type *int*

XtVaNestedList
: This is followed by one additional argument of type *XtVarArgsList*. This argument is a nested list of *varargs* returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateMenuBar(3), **XmCreateRowColumn**(3), **XmCreateSimpleMenuBar**(3), and **XmRowColumn**(3).

XmVaCreateSimpleOptionMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmVaCreateSimpleOptionMenu(
  Widget *parent***,**
  String *name***,**
  XmString *option_label***,**
  KeySym *option_mnemonic***,**
  int *button_set***,**
  XtCallbackProc *callback***);**

Description

XmVaCreateSimpleOptionMenu creates an instance of a RowColumn widget of type **XmMENU_OPTION** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (*varargs*) calling convention.

This routine creates an OptionMenu and its Pulldown submenu containing PushButtonGadget or CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to the number of separators in the menu minus 1. Buttons and separators are named and created in the order in which they are specified in the variable portion of the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
option_label	Specifies the label string to be used on the left side of the OptionMenu.
option_mnemonic	Specifies a keysym for a key that, when pressed by the user, posts the associated Pulldown menu pane.

button_set Specifies which PushButtonGadget is initially set. The value is the integer *n* that corresponds to the *n*th PushButtonGadget specified in the variable portion of the argument list. Only a PushButtonGadget can be set, and only PushButtonGadgets are counted in determining the integer *n*. The first PushButtonGadget is number 0 (zero).

callback Specifies a callback procedure to be called when a button is activated. This callback function is added to each button after creation as the button's **XmNactivateCallback**. The callback function is called when a button is activated, and the button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of *varargs*:

XmVaPUSHBUTTON
This is followed by four additional arguments. The set specifies one button in the OptionMenu's Pulldown submenu and some of its resource values. The button created is a PushButtonGadget. Following are the additional four arguments, in order:

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**
accelerator	The accelerator, of type **String**
accelerator_text	The accelerator text, of type **XmString**

XmVaSEPARATOR
This is followed by no additional arguments. It specifies one separator in the OptionMenu's Pulldown submenu.

XmVaDOUBLE_SEPARATOR
This is followed by no additional arguments. It specifies one separator in the OptionMenu's Pulldown submenu. The separator type is **XmDOUBLE_LINE**.

resource_name
This is followed by one additional argument, the value of the resource, of type *XtArgVal*. The pair specifies a resource and its value for the Pulldown submenu.

XtVaTypedArg

This is followed by four additional arguments. The set specifies a resource and its value for the Pulldown submenu. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type *XtArgVal*
size	The size of the resource value in bytes, of type *int*

XtVaNestedList

This is followed by one additional argument of type *XtVarArgsList*. This argument is a nested list of *varargs* returned by **XtVaCreateArgsList**.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas:

Option Menu Label Gadget
OptionLabel

Option Menu Cascade Button
OptionButton

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateOptionMenu(3), **XmCreateRowColumn**(3), **XmCreateSimpleOptionMenu**(3), and **XmRowColumn**(3).

XmVaCreateSimplePopupMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmVaCreateSimplePopupMenu(
    Widget *parent***,**
    String *name***,**
    XtCallbackProc *callback***);**

Description

XmVaCreateSimplePopupMenu creates an instance of a RowColumn widget of type **XmMENU_POPUP** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (*varargs*) calling convention.

This routine creates a Popup menu pane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to the number of separators in the menu minus 1. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to the number of titles in the menu minus 1. Buttons, separators, and titles are named and created in the order in which they are specified in the variable portion of the argument list.

parent	Specifies the widget ID of the parent of the MenuShell
name	Specifies the name of the created widget
callback	Specifies a callback procedure to be called when a button is activated or when its value changes. This callback function is added to each button after creation. For a CascadeButtonGadget or a PushButtonGadget, the callback is added as the button's **XmNactivateCallback**, and it is called when the button is activated. For a ToggleButtonGadget, the callback is added as the button's **XmNvalueChangedCallback**, and it is called when the button's value changes. The button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. The following list describes the possible first arguments in each group of *varargs*.

XmVaCASCADEBUTTON

This is followed by two additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a CascadeButtonGadget. Following are the additional two arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

XmVaPUSHBUTTON

This is followed by four additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a PushButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaRADIOBUTTON

This is followed by four additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaCHECKBUTTON

This is followed by four additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**
accelerator	The accelerator, of type **String**
accelerator_text	The accelerator text, of type **XmString**

XmVaTITLE

This is followed by one additional argument. The pair specifies a title LabelGadget in the PopupMenu. Following is the additional argument:

title	The title string, of type **XmString**

XmVaSEPARATOR

This is followed by no additional arguments. It specifies one separator in the PopupMenu.

XmVaDOUBLE_SEPARATOR

This is followed by no additional arguments. It specifies one separator in the PopupMenu. The separator type is **XmDOUBLE_LINE**.

resource_name

This is followed by one additional argument, the value of the resource, of type *XtArgVal*. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg

This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**

value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type *XtArgVal*
size	The size of the resource value in bytes, of type *int*

XtVaNestedList
This is followed by one additional argument of type *XtVarArgsList*. This argument is a nested list of *varargs* returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreatePopupMenu(3), **XmCreateRowColumn**(3), **XmCreateSimplePopupMenu**(3), and **XmRowColumn**(3).

XmVaCreateSimplePulldownMenu

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmVaCreateSimplePulldownMenu(
 Widget *parent***,**
 String *name***,**
 int *post_from_button***,**
 XtCallbackProc *callback***);**

Description

XmVaCreateSimplePulldownMenu creates an instance of a RowColumn widget of type **XmMENU_PULLDOWN** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (*varargs*) calling convention.

This routine creates a Pulldown menu pane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 to the number of buttons in the menu minus 1. The name of each separator is **separator_***n*, where *n* is an integer from 0 to the number of separators in the menu minus 1. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to the number of titles in the menu minus 1. Buttons, separators, and titles are named and created in the order in which they are specified in the variable portion of the argument list.

This routine also attaches the PulldownMenu to a CascadeButton or CascadeButtonGadget in the parent. The PulldownMenu is then posted from this button.

parent Specifies the widget ID of the parent of the MenuShell.

name Specifies the name of the created widget.

post_from_button
Specifies the CascadeButton or CascadeButtonGadget in the parent to which the Pulldown menu pane is attached. The value is the integer *n* that

corresponds to the *n*th CascadeButton or CascadeButtonGadget specified for the parent of the Pulldown menu pane. A Pulldown menu pane can be attached only to a CascadeButton or CascadeButtonGadget, and only CascadeButtons and CascadeButtonGadgets are counted in determining the integer *n*. The first CascadeButton or CascadeButtonGadget is number 0 (zero).

callback Specifies a callback procedure to be called when a button is activated or when its value changes. This callback function is added to each button after creation. For a CascadeButtonGadget or a PushButtonGadget, the callback is added as the button's **XmNactivateCallback**, and it is called when the button is activated. For a ToggleButtonGadget, the callback is added as the button's **XmNvalueChangedCallback**, and it is called when the button's value changes. The button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of *varargs*:

XmVaCASCADEBUTTON
This is followed by two additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a CascadeButtonGadget. Following are the additional two arguments, in order:

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**

XmVaPUSHBUTTON
This is followed by four additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a PushButtonGadget. Following are the additional four arguments, in order:

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**
accelerator	The accelerator, of type **String**
accelerator_text	The accelerator text, of type **XmString**

XmVaRADIOBUTTON

This is followed by four additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label	The label string, of type **XmString**
mnemonic	The mnemonic, of type **KeySym**
accelerator	The accelerator, of type **String**
accelerator_text	The accelerator text, of type **XmString**

XmVaCHECKBUTTON

This is followed by four additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label	The label string, of type **XmString.**
mnemonic	The mnemonic, of type **KeySym**
accelerator	The accelerator, of type **String**
accelerator_text	The accelerator text, of type **XmString**

XmVaTITLE

This is followed by one additional argument. The pair specifies a title LabelGadget in the PulldownMenu. Following is the additional argument:

title	The title string, of type **XmString**

XmVaSEPARATOR

This is followed by no additional arguments. It specifies one separator in the PulldownMenu.

XmVaDOUBLE_SEPARATOR

This is followed by no additional arguments. It specifies one separator in the PulldownMenu. The separator type is **XmDOUBLE_LINE**.

resource_name

This is followed by one additional argument, the value of the resource, of type XtArgVal. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg

This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type String.
type	The type of the resource value supplied, of type String.
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type XtArgVal.
size	The size of the resource value in bytes, of type int.

XtVaNestedList

This is followed by one additional argument of type XtVarArgsList. This argument is a nested list of *varargs* returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreatePulldownMenu(3), **XmCreateRowColumn**(3), **XmCreateSimplePulldownMenu**, and **XmRowColumn**(3).

XmVaCreateSimpleRadioBox

Purpose

A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmVaCreateSimpleRadioBox(
    Widget *parent***,**
    String *name***,**
    int *button_set***,**
    XtCallbackProc *callback***);**

Description

XmVaCreateSimpleRadioBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (*varargs*) calling convention.

This routine creates a RadioBox and its ToggleButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to the number of buttons in the menu minus 1.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
button_set	Specifies which button is initially set. The value is the integer *n* in the button name **button_***n*.
callback	Specifies a callback procedure to be called when a button's value changes. This callback function is added to each button after creation as the button's **XmNvalueChangedCallback**. The callback function is called when a button's value changes, and the button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of *varargs*:

XmVaRADIOBUTTON
This is followed by four additional arguments. The set specifies one button in the RadioBox and some of its resource values. Following are the additional four arguments, in order:

label	The label string, of type **XmString**.
mnemonic	The mnemonic, of type **KeySym**. This is ignored in this release.
accelerator	The accelerator, of type **String**. This is ignored in this release.
accelerator_text	The accelerator text, of type **XmString**. This is ignored in this release.

resource_name
This is followed by one additional argument, the value of the resource, of type *XtArgVal*. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in this order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type *XtArgVal*
size	The size of the resource value in bytes, of type *int*

XtVaNestedList
This is followed by one additional argument of type *XtVarArgsList*. This argument is a nested list of *varargs* returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn**(3).

Return Values

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3), **XmCreateRowColumn**(3), **XmCreateSimpleCheckBox**(3), **XmCreateSimpleRadioBox**(3), **XmRowColumn**(3), and **XmVaCreateSimpleCheckBox**(3),

XmWidgetGetBaselines

Purpose

Retrieves baseline information for a widget

Synopsis

#include <Xm/Xm.h>

Boolean XmWidgetGetBaselines(
Widget *widget***,**
Dimension *****baselines*,**
int ****line_count*);**

Description

XmWidgetGetBaselines returns an array that contains one or more baseline values associated with the specified widget. The baseline of any given line of text is a vertical offset in pixels from the origin of the widget’s bounding box to the given baseline.

widget	Specifies the ID of the widget for which baseline values are requested
baselines	Returns an array that contains the value of each baseline of text in the widget. The function allocates space to hold the returned array. The application is responsible for managing the allocated space. The application can recover this allocated space by calling **XtFree**.
line_count	Returns the number of lines in the widget

Return Values

Returns a Boolean value that indicates whether the widget contains a baseline. If the value is True, the function returns a value for each baseline of text. If it is False, the function was unable to return a baseline value.

Related Information

XmWidgetGetDisplayRect(3).

XmWidgetGetDisplayRect

Purpose

Retrieves display rectangle information for a widget

Synopsis

#include <Xm/Xm.h>

Boolean XmWidgetGetDisplayRect(
  Widget *widget***,**
  XRectangle ****displayrect*);**

Description

XmWidgetGetDisplayRect returns the width, height and the x and y-coordinates of the upper left corner of the display rectangle of the specified widget. The display rectangle is the smallest rectangle that encloses either a string or a pixmap.

If the widget contains a string, the return values specify the x and y-coordinates of the upper left corner of the display rectangle relative to the origin of the widget and the width and height in pixels.

In the case of a pixmap, the return values specify the x and y-coordinates of the upper left corner of the pixmap relative to the origin, and the width and height of the pixmap in pixels.

widget	Specifies the widget ID
displayrect	Specifies a pointer to an XRectangle structure in which the x and y-coordinates, width and height of the display rectangle are returned

Return Values

Returns True if the specified widget has an associated display rectangle; otherwise, returns False.

Related Information

XmWidgetGetBaselines(3).