X Python Reference Manual

Sjoerd Mullender
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
E-mail: `sjoerd@cwi.nl`

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

25 October 1996
Release 1.4

Abstract

This document describes the built-in types, exceptions, and functions of the X windows extension to Python. It assumes basic knowledge about the Python language and access to the X windows documentation. For an informal introduction to the language, see the *Python Tutorial*. The *Python Reference Manual* gives a more formal definition of the language. The *Python Library Reference* describes the built-in and standard modules of Python. This document can be seen as en extension to that document.
CR Subject Classification (1991): D.3.2, D.3.3, D.1.5, D.2.2, I.3.4.
Keywords & Phrases: Object-oriented languages, Python, libraries, modules, Spanish Inquisition, SPAM.

Table of Contents

Chapter 1
Introduction

This document describes the calling sequences and return values of all functions that have been implemented in the X extension of Python. If a function is not mentioned in this document, it was not implemented. In addition, the actual X function that is called internally is mentioned. For the precise semantics of the function, please refer to the X documentation.

In many cases, functions in the X libraries have been turned into methods of various objects. See the sections on the various objects for lists of defined methods. See the index to find of which object a function is a method.

As an appetizer, here is the Hello World program using Motif.

```
import sys, Xt, Xm

def main():
    toplevel = Xt.Initialize('XHello', [], sys.argv)
    hello = toplevel.CreateManagedWidget('hello', Xm.Label,
                                         {'labelString': 'Hello World'})
    toplevel.RealizeWidget()
    Xt.MainLoop()

main()
```

1. Naming and coding conventions

Prefixes, such as `Xt`, `Xm`, and `Glx` are dropped from a function name.

Only Xt functions that have no widget or widget class argument are defined directly in the Xt module.

`XtInitialize` must be called as `Xt.Initialize('classname', [], sys.argv)` or as `Xt.Initialize()`.

Xt and Xm functions with a widget parameter are changed into widget methods, with the widget parameter removed, even if it is not the first parameter.

Xt functions with a widget class parameter are changed into widget class methods.

Parameters, return values, and resources of type `XmString` are replaced by ordinary Python strings.

Parameters representing structures are replaced by Python tuples.

Some functions accept a partially filled struct with a flag describing which members are filled in. This is represented by a dictionary where the key is the name of the struct members to be filled in, and the value the value of these members. No flag is used.

Pairs of parameters describing arrays of strings or structures are replaced by a Python list of strings or tuples.

Output parameters are always replaced by Python function return values. If there is more than one output parameter, a tuple is constructed. If the functions also returns a value, it is the first element of the tuple.

Input parameters of type `ArgList`, such as those given to `XtCreateWidget` and `XtSetValues`, are replaced by a dictionary with the resource name as key and the resource value as value. Also, the return value of `XtGetValues` is such a dictionary.

Functions in Xlib that take both a Display and a Window argument are implemented as widget methods (if they are implemented at all). The Display and Window arguments are then removed from the argument list.

2. Drawing

Python Xt supports drawing operations, usable e.g. in Motif DrawingArea widgets and on off-screen Pixmaps. In general the X man pages should be consulted for the precise semantics.

An off-screen Pixmap can be created using the widget method `CreatePixmap`.

In order to draw in a widget or on a Pixmap, a Graphics Context object instance must be created. This can be done using the widget methods `CreateGC` and `GetGC`, or the Pixmap method `CreateGC`.

Once a Graphics Context object is instantiated, the Xlib drawing functions can be used. These drawing functions are methods of the Graphics Context object.

For example, in order to draw a rectangle in a window, the following code could be used.

```
values = {'line_style': X.LineOnOffDash}
gc = w.GetGC(values)
gc.DrawRectangle(x, y, width, height)
```

2.1 Drawing Images

An image (pixmap) can be drawn by using the GC method `PutImage`. This method requires an Image object which can be created using the Visual Info method `CreateImage`. The only way to get a Visual Info object is by using the widget method `GetVisualInfo` which returns a list of Visual Info objects.

For example (`picture` is a string object that contains an image in the correct format for the call to CreateImage, see the manual for more information):

```
visual = w.DefaultVisualOfScreen()
image = visual.CreateImage(depth, X.ZPixmap, 0, picture, im_w, im_h,
                           pad, 0)
gc.PutImage(image, im_x, im_y, win_x, win_y, win_w, win_h)
```

2.2 Drawing Text

To draw text, a font must first be selected. A font object is acquired by using the widget method `LoadQueryFont`. This object can then be passed to the GC method `SetFont` or assigned to the GC instance variable `font`.

For example:

```
font = w.LoadQueryFont(
                '-*-times-medium-r-*-*-*-100-*-*-*-*-iso8859-1')
gc.SetFont(font)
gc.DrawString(x, y, 'Hello World!')
```

3. Non-default visuals

First you must find the visual you want to use, then you must create a colormap using that visual. When creating a top-level shell widget, you must specify the visual, the colormap, and the depth of the visual.

Motif only allows you to change the visual of a shell widget, so the visual and colormap must be created before the shell widget is created.

When creating another shell widget, the new widget inherits the colormap and depth from the parent widget, but it inherits the visual from the parent window (the root window). Thus on every shell you create you must specify visual, colormap, and depth. Note that popup dialogs and menus are also shells.

Sample code that searches for a visual and creates a colormap which is then used by a shell widget is given here. (Note that `Xm` must be imported for the `CreateForm` method to be available.)

```
import Xt, Xm, X, sys
Xt.ToolkitInitialize()
dpy = Xt.OpenDisplay(None, None, None, [], sys.argv)
visuals = dpy.GetVisualInfo({'class': X.TrueColor})
```

```
if not visuals:
    print 'No TrueColor visuals available'
    sys.exit(1)
v_best = visuals[0]
for v in visuals:
    if v.depth > v_best.depth:
        v_best = v
colormap = v_best.CreateColormap(X.AllocNone)
toplevel = Xt.CreateApplicationShell('shell', Xt.ApplicationShell,
    {'visual': v_best, 'depth': v_best.depth, 'colormap': colormap})
form = toplevel.CreateForm('form', {'width': 100, 'height': 100})
form.ManageChild()
toplevel.RealizeWidget()
Xt.MainLoop()
```

4. Callback data and events

The following table lists the functions and methods that install callback functions together with the calling sequence of the callback function.

```
Xt.AddInput:                   callback(client_data, fileno, id)
Xt.AddTimeOut:                 callback(client_data, id)
Xt.AddWorkProc:                callback(client_data)
Xt.AddEventHandler:            continue_to_dispatch = callback(w,
                                       client_data, call_data)
Xt.AddRawEventHandler:         continue_to_dispatch = callback(w,
                                       client_data, call_data)
Xt.AddActions:                 callback(w, call_data, plist)
Xt.AddActionHook:              callback(w, client_data, action,
                                        call_data, plist)
w.AddCallback:                 callback(w, client_data, call_data)
insertPosition resource:       position = callback(w)
createPopupChildProc resource: callback(w)
dropProc resource:             callback(w, None, call_data)
dragProc resource:             callback(w, None, call_data)
transferProc resource:         callback(w, client_data, selection,
                                        type, value, length, format)
convertProc resource:          type, value, format = callback(w,
                                        selection, target, max_length,
                                        None, request_id)
```

Here, `client_data` is the user-supplied data, `fileno` is the file descriptor that was being checked, `id` is the identifier that `Xt.AddInput` or `Xt.AddTimeOut` returned, `w` is the widget object on behalf of which the callback is called or installed, and `call_data` is the data giving the reason for the callback which the library supplies. If the callback function for an event handler returns 0, the `continue_to_dispatch` return parameter of the library function is cleared. See the manual pages for `XtAddEventHandler` and `XtAddRawEventHandler`.

The `call_data` parameter is an object representing the structure that is passed by the library, possibly ultimately by the user through the widget method `CallCallbacks`. The object has attributes with the same names as the fields of the structure it represents. Many attributes can be changed, and the change is reflected in the underlying structure. This is especially useful for the `XmTextField` and `XmText` widgets.

The `call_data` parameter of the event and action handlers is always an object representing a `XEvent` structure. The structure that is passed to a callback installed with the `AddCallback` method depends on the class of the widget and the type of the callback. For Athena widgets, the structure is a `XEvent`. For Motif widgets, the structure contains at least the fields `reason` and `event`. The `event` is a `XEvent` structure. Which attributes exist can be seen by using the `__members__` attribute which returns a list of attribute names. The name of the structure is given by the `__name__` attribute.

The attributes of the `call_data` parameter are only valid as long as the callback function is active. It is useless to keep a reference to the object for use after the callback function returns. Also, the object returned by a reference to the `event` attribute of a Motif callback is invalid when the callback is not active anymore. Access to attributes of these objects result in an exception.

Here is an example of a Motif TextField widget that uses a verification callback to make sure only digits are entered. Accepted digits are changed to '*', so this code can be used as the basis for a function that reads personal identification numbers.

```
import Xt, Xm

def cbmodify(w, client_data, call_data):
    if call_data.text:
        # only accept digits...
        for c in call_data.text:
            if c not in '0123456789':
                call_data.doit = 0
        # ... and change them into *'s
        call_data.text = '*' * len(call_data.text)

top = Xt.Initialize()
text = top.CreateManagedWidget('text', Xm.TextField, {})
text.AddCallback('modifyVerifyCallback', cbmodify, None)
top.RealizeWidget()
Xt.MainLoop()
```

Some functions and methods return or accepts as parameters objects representing `XEvent` structures. These objects are compatible with the `call_data` objects described above. Event objects that are returned by functions and methods remain valid as long as there is a reference to them.

5. Unsupported features

None of the `XtApp*` functions have been implemented, but most of the corresponding functions without `App` have been implemented, except where noted below. Thus you cannot use an other than the default application context. Other unimplemented functions are `XtCreateApplicationContext` and `XtDestroyApplicationContext`.

The functions that deal with application resources have not been implemented. The affected functions are: `XtGetApplicationResources`, `XtAddConverter`, and all `Xrm*` functions. A workaround is to use `void = os.system('xrdb -nocpp -merge <name>.resources')`.

The geometry management functions `XtQueryGeometry` and `XtMakeResizeRequest` are not implemented.

Only a few functions with Display, Screen, or Event arguments or return types have been implemented.

The `XmClipboard*` functions have not been implemented.

The `XmCvt*` functions have not been implemented.

The Motif Pixmap and Image functions (`XmDestroyPixmap`, `XmInstallImage`, `XmUninstallImage`, and `XmGetPixmap`) have not been implemented.

The `XmFontList*` functions have not been implemented.

Chapter 2
Built-in module Xlib

1. Xlib functions and variables

The module defines the following functions and variables:

`Error`
: Exception generated by functions in this module.

`SpecificationRelease`
: XlibSpecificationRelease.

`CreateRegion()`
: Returns `region_object`. This function corresponds with `XCreateRegion`.

`Event(`*type* [, *argdict*] `)`
: Returns `event`.

`InitThreads()`
: Returns `status`. This function corresponds with `XInitThreads`.

`LookupString(`*event*`)`
: Returns `(string, keysym)`. This function corresponds with `XLookupString`.

`PolygonRegion(`*XPoint[]*, *fill_rule*`)`
: Returns `region_object`. This function corresponds with `XPolygonRegion`.

2. Display object

A display object can only be acquired by using the widget method `Display` or by the `Xt` function `OpenDisplay`.

Display objects (display_objects) have the following methods:

`Bell(`*percent*`)`
: Returns `None`. This function corresponds with `XBell`.

`CheckMaskEvent(`*event_mask*`)`
: Returns `None`. This function corresponds with `XCheckMaskEvent`.

`ConnectionNumber()`

Returns `connection_number`. This function corresponds with `ConnectionNumber`.

`CreateFontCursor(`*shape*`)`
: Returns `cursor`. This function corresponds with `XCreateFontCursor`.

`Flush()`
: Returns `None`. This function corresponds with `XFlush`.

`GetAtomName(`*atom*`)`
: Returns `atom_name`. This function corresponds with `XGetAtomName`.

`GetVisualInfo(`*template_dict*`)`
: Returns `vi_object_list`. This function corresponds with `XGetVisualInfo`.

`InternAtom(`*atom_name*`, `*only_if_exists*`)`
: Returns `atom`. This function corresponds with `XInternAtom`.

`LockDisplay()`
: Returns `None`. This function corresponds with `XLockDisplay`.

`NextEvent()`
: Returns `event`. This function corresponds with `XNextEvent`.

`PeekEvent()`
: Returns `event`. This function corresponds with `XPeekEvent`.

`PutBackEvent(`*event*`)`
: Returns `None`. This function corresponds with `XPutBackEvent`.

`SendEvent(`*window*`, `*propagate*`, `*event_mask*`, `*event*`)`
: Returns `success`. This function corresponds with `XSendEvent`.

`Synchronize(`*onoff*`)`
: Returns `None`. This function corresponds with `XSynchronize`.

`UnlockDisplay()`
: Returns `None`. This function corresponds with `XUnlockDisplay`.

3. Region object

A region object can only be acquired by using the `Xlib` function `CreateRegion`.
Some of the Region functions in the X library have a result argument. In the region object methods, the methods modify the object of which the method is called instead.
Region objects (region_objects) have the following methods:

`AddExposureToRegion(`*event*`)`
: Returns `None`. This function corresponds with `XtAddExposureToRegion`.

`ClipBox()`
: Returns `(x, y, width, height)`. This function corresponds with `XClipBox`.

`EmptyRegion()`
: Returns `success`. This function corresponds with `XEmptyRegion`.

`EqualRegion(`*region*`)`
: Returns `success`. This function corresponds with `XEqualRegion`.

`IntersectRegion(`*region*`)`
: Returns `None`. This function corresponds with `XIntersectRegion`.

`OffsetRegion(`*dx*`, `*dy*`)`
: Returns `None`. This function corresponds with `XOffsetRegion`.

`PointInRegion(`*x*`, `*y*`)`
: Returns `success`. This function corresponds with `XPointInRegion`.

`RectInRegion(`*x*`,` *y*`,` *width*`,` *height*`)`
  Returns `success`. This function corresponds with `XRectInRegion`.

`ShrinkRegion(`*dx*`,` *dy*`)`
  Returns `None`. This function corresponds with `XShrinkRegion`.

`SubtractRegion(`*region*`)`
  Returns `None`. This function corresponds with `XSubtractRegion`.

`UnionRectWithRegion(`*x*`,` *y*`,` *width*`,` *height*`)`
  Returns `None`. This function corresponds with `XUnionRectWithRegion`.

`UnionRegion(`*region*`)`
  Returns `None`. This function corresponds with `XUnionRegion`.

`XorRegion(`*region*`)`
  Returns `None`. This function corresponds with `XXorRegion`.

Chapter 3
Built-in module Xt

1. Xt functions and variables

The functions in the Xt library that have a widget argument are all turned into widget methods. The functions in the Xt library that have a widget class argument have been turned into widget_class methods. The remaining functions are given here.

The module defines the following functions and variables:

`ApplicationShell`
: Widget class object.

`Composite`
: Widget class object.

`Constraint`
: Widget class object.

`OverrideShell`
: Widget class object.

`Shell`
: Widget class object.

`SpecificationRelease`
: XtSpecificationRelease.

`TopLevelShell`
: Widget class object.

`TransientShell`
: Widget class object.

`VendorShell`
: Widget class object.

`WMShell`
: Widget class object.

`AddActionHook(`*cbfunc*`,` *cbarg*`)`
: Returns `action_id`. This function corresponds with `XtAppAddActionHook`.

`AddActions(`*argdict*`)`
: Returns `None`. This function corresponds with `XtAddActions`.

`AddInput(`*source*`,` *condition*`,` *proc*`,` *client_data*`)`
: Returns `input_id`. This function corresponds with `XtAddInput`.

`AddTimeOut(`*interval*`,` *proc*`,` *client_data*`)`
: Returns `interval_id`. This function corresponds with `XtAddTimeOut`.

`AddWorkProc(`*proc*`,` *client_data*`)`
: Returns `workproc_id`. This function corresponds with `XtAddWorkProc`.

`AppInitialize(`*app_class*`,` *option_list*`,` *fallback_resource_list*`,` *argdict*`)`
: Returns `widget`. This function corresponds with `XtAppInitialize`.

`CreateApplicationShell(`*application_name*`,` *widget_class*`,` *argdict*`)`
: Returns `widget`. This function corresponds with `XtCreateApplicationShell`.

`DispatchEvent(`*event*`)`
: Returns `success`. This function corresponds with `XtDispatchEvent`.

`Error`
: Exception generated by functions in this module.

`GetSelectionTimeout()`
: Returns `timeout`. This function corresponds with `XtGetSelectionTimeout`.

`Initialize(` [*classname*`,` *option_list*`,` *argv*] `)`
: Returns `widget`. This function corresponds with `XtInitialize`.

`MainLoop()`
: Returns `None`. This function corresponds with `XtMainLoop`.

`ManageChildren(`*widget_list*`)`
: Returns `None`. This function corresponds with `XtManageChildren`.

`NextEvent()`
: Returns `event`. This function corresponds with `XtNextEvent`.

`OpenDisplay(` [*display*`,` *appname*`,` *classname*`,` *option_list*`,` *argv*] `)`
: Returns `display_object`. This function corresponds with `XtOpenDisplay`.

`ParseTranslationTable(`*table*`)`
: Returns `translation_object`. This function corresponds with `XtParseTranslationTable`.

`PeekEvent()`
: Returns `event` or `None`. This function corresponds with `XtPeekEvent`.

`Pending()`
: Returns `mask`. This function corresponds with `XtPending`.

`ProcessEvent(`*mask*`)`
: Returns `None`. This function corresponds with `XtProcessEvent`.

`RemoveActionHook(`*action_id*`)`
: Returns `None`. This function corresponds with `XtRemoveActionHook`.

`RemoveInput(`*input_id*`)`
: Returns `None`. This function corresponds with `XtRemoveInput`.

`RemoveTimeOut(`*interval_id*`)`
: Returns `None`. This function corresponds with `XtRemoveTimeOut`.

`RemoveWorkProc(`*workproc_id*`)`
: Returns `None`. This function corresponds with `XtRemoveWorkProc`.

`SetFallbackResources(`*resource_list*`)`
Returns `None`. This function corresponds with `XtAppSetFallbackResources`.

`SetSelectionTimeout(`*timeout*`)`
Returns `None`. This function corresponds with `XtSetSelectionTimeout`.

`ToolkitInitialize()`
Returns `None`. This function corresponds with `XtToolkitInitialize`.

2. Widget Class object

Widget Class objects have the following methods:

`GetConstraintResourceList()`
Returns `resource_list`. This function corresponds with `XtGetConstraintResourceList`.

`GetResourceList()`
Returns `resource_list`. This function corresponds with `XtGetResourceList`.

`InitializeWidgetClass()`
Returns `None`. This function corresponds with `XtInitializeWidgetClass`.

3. Widget object

Widget objects have the following methods:

`AddCallback(`*callback_name*`, `*callback*`, `*client_data*`)`
Returns `None`. This function corresponds with `XtAddCallback`.

`AddEventHandler(`*event_mask*`, `*nonmaskable*`, `*proc*`, `*client_data*`)`
Returns `None`. This function corresponds with `XtAddEventHandler`.

`AddGrab(`*exclusive*`, `*spring_loaded*`)`
Returns `None`. This function corresponds with `XtAddGrab`.

`AddRawEventHandler(`*event_mask*`, `*nonmaskable*`, `*proc*`, `*client_data*`)`
Returns `None`. This function corresponds with `XtAddRawEventHandler`.

`AugmentTranslations(`*translation_object*`)`
Returns `None`. This function corresponds with `XtAugmentTranslations`.

`BuildEventMask()`
Returns `event_mask`. This function corresponds with `XtBuildEventMask`.

`CallCallbacks(`*callback_name*`, `*call_data*`)`
Returns `None`. This function corresponds with `XtCallCallbacks`.

`CancelSelectionRequest(`*selection*`)`
Returns `None`. This function corresponds with `XtCancelSelectionRequest`.

`ChangeWindowAttributes(`*window_attribute_dict*`)`
Returns `None`. This function corresponds with `XChangeWindowAttributes`.

`Class()`
Returns `widget_class`. This function corresponds with `XtClass`.

`Convert(`*from_value*`, `*to_type*`)`
Returns `to_value`. This function corresponds with `XtConvertAndStore or XtConvert`.

`CreateColormap(`*vi_object*`, `*alloc*`)`
Returns `cm_object`. This function corresponds with `XCreateColormap`.

`CreateGC(`*gc_dict*`)`
: Returns `gc_object`. This function corresponds with `XCreateGC`.

`CreateManagedWidget(`*name*, *widget_class*, *argdict*`)`
: Returns `widget`. This function corresponds with `XtCreateManagedWidget`.

`CreatePixmap(` [*width*, *height* [, *depth*]] `)`
: Returns `pixmap_object`. This function corresponds with `XCreatePixmap`.

`CreatePopupShell(`*name*, *widget_class*, *argdict*`)`
: Returns `widget`. This function corresponds with `XtCreatePopupShell`.

`CreateSelectionRequest(`*selection*`)`
: Returns `None`. This function corresponds with `XtCreateSelectionRequest`.

`CreateWidget(`*name*, *widget_class*, *argdict*`)`
: Returns `widget`. This function corresponds with `XtCreateWidget`.

`DefaultColormap()`
: Returns `cm_object`. This function corresponds with `DefaultColormapOfScreen`.

`DefaultColormapOfScreen()`
: Returns `cm_object`. This function corresponds with `DefaultColormapOfScreen`.

`DefaultVisual()`
: Returns `vi_object`. This function corresponds with `DefaultVisualOfScreen`.

`DefaultVisualOfScreen()`
: Returns `vi_object`. This function corresponds with `DefaultVisualOfScreen`.

`DefineCursor(`*cursor*`)`
: Returns `None`. This function corresponds with `XDefineCursor`.

`DestroyWidget()`
: Returns `None`. This function corresponds with `XtDestroyWidget`.

`DisownSelection(`*selection*, *time*`)`
: Returns `None`. This function corresponds with `XtDisownSelection`.

`Display()`
: Returns `display_object`. This function corresponds with `XtDisplay`.

`GetGC(`*gc_dict*`)`
: Returns `gc_object`. This function corresponds with `XtGetGC`.

`GetGeometry()`
: Returns `(root, x, y, width, height, border_width, depth)`. This function corresponds with `XGetGeometry`.

`GetSelectionParameters(`*selection*, *request_id*`)`
: Returns `(type, value, length, format)` or `None`. This function corresponds with `XtGetSelectionParameters`.

`GetSelectionRequest(`*selection*, *request_id*`)`
: Returns `event`. This function corresponds with `XtGetSelectionRequest`.

`GetSelectionValue(`*selection*, *target*, *callback*, *client_data*, *time*`)`
: Returns `None`. This function corresponds with `XtGetSelectionValue`.

`GetSelectionValueIncremental(`*selection*, *target*, *callback*, *client_data*, *time*`)`
: Returns `None`. This function corresponds with `XtGetSelectionValueIncremental`.

`GetSelectionValues(`*selection*, *target* [], *callback*, *client_data*[] , *time*`)`
: Returns `None`. This function corresponds with `XtGetSelectionValues`.

`GetSelectionValuesIncremental`(*selection*, *target* [], *callback*, *client_data*[] , *time*)
Returns `None`. This function corresponds with `XtGetSelectionValuesIncremental`.

`GetValues`(*name_list*)
Returns `argdict`. This function corresponds with `XtGetValues`.

`GetVisualInfo`(*template_dict*)
Returns `vi_object_list`. This function corresponds with `XGetVisualInfo`.

`GetWMNormalHints`()
Returns `hintdict`. This function corresponds with `XGetWMNormalHints`.

`GrabButton`(*button*, *modifiers*, *owner_events*, *event_mask*, *pointer_mode*, *keyboard_mode*, *confine_to_widget*, *cursor*)
Returns `None`. This function corresponds with `XGrabButton`.

`HasCallbacks`(*callback_name*)
Returns `callback_status`. This function corresponds with `XtHasCallbacks`.

`HeightMMOfScreen`()
Returns `height_mm`. This function corresponds with `HeightMMOfScreen`.

`HeightOfScreen`()
Returns `height`. This function corresponds with `HeightOfScreen`.

`InstallAccelerators`(*source*)
Returns `None`. This function corresponds with `XtInstallAccelerators`.

`InstallAllAccelerators`(*source*)
Returns `None`. This function corresponds with `XtInstallAllAccelerators`.

`IsComposite`()
Returns `success`. This function corresponds with `XtIsComposite`.

`IsManaged`()
Returns `success`. This function corresponds with `XtIsManaged`.

`IsRealized`()
Returns `success`. This function corresponds with `XtIsRealized`.

`IsSensitive`()
Returns `success`. This function corresponds with `XtIsSensitive`.

`IsSubclass`(*widget_class*)
Returns `success`. This function corresponds with `XtIsSubclass`.

`ListFonts`(*pattern*)
Returns `font_list`. This function corresponds with `XListFonts`.

`LoadQueryFont`(*name*)
Returns `font_object`. This function corresponds with `XLoadQueryFont`.

`LowerWindow`()
Returns `None`. This function corresponds with `XLowerWindow`.

`ManageChild`()
Returns `None`. This function corresponds with `XtManageChild`.

`MapWidget`()
Returns `None`. This function corresponds with `XtMapWidget`.

`Name`()
Returns `None`. This function corresponds with `XtName`.

`NameToWidget`(*name*)
Returns `widget`. This function corresponds with `XtNameToWidget`.

`OverrideTranslations`(*translation_object*)
  Returns `None`. This function corresponds with `XtOverrideTranslations`.

`OwnSelection`(*selection*, *time*, *convert_proc*, *lose_selection*, *done_proc*)
  Returns `success`. This function corresponds with `XtOwnSelection`.

`Parent()`
  Returns `parent_widget`. This function corresponds with `XtParent`.

`Popdown()`
  Returns `None`. This function corresponds with `XtPopdown`.

`Popup`(*grab_kind*)
  Returns `None`. This function corresponds with `XtPopup`.

`QueryPointer()`
  Returns `(root, child, root_x, root_y, win_x, win_y, mask)`. This function corresponds with `XQueryPointer`.

`RaiseWindow()`
  Returns `None`. This function corresponds with `XRaiseWindow`.

`ReadBitmapFile`(*file*)
  Returns `(width, height, pixmap_object, hotspot_x, hotspot_y)`. This function corresponds with `XReadBitmapFile`.

`RealizeWidget()`
  Returns `None`. This function corresponds with `XtRealizeWidget`.

`RemoveAllCallbacks`(*callback_name*)
  Returns `None`. This function corresponds with `XtRemoveAllCallbacks`.

`RemoveCallback`(*callback_name*, *callback*, *client_data*)
  Returns `None`. This function corresponds with `XtRemoveAllCallbacks`.

`RemoveEventHandler`(*event_mask*, *nonmaskable*, *proc*, *client_data*)
  Returns `None`. This function corresponds with `XtRemoveEventHandler`.

`RemoveGrab()`
  Returns `None`. This function corresponds with `XtRemoveGrab`.

`RemoveRawEventHandler`(*event_mask*, *nonmaskable*, *proc*, *client_data*)
  Returns `None`. This function corresponds with `XtRemoveRawEventHandler`.

`SendSelectionRequest`(*selection*, *time*)
  Returns `None`. This function corresponds with `XtSendSelectionRequest`.

`SetMappedWhenManaged`(*map_when_managed*)
  Returns `None`. This function corresponds with `XtSetMappedWhenManaged`.

`SetSelectionParameters`(*selection*, *type*, *value*, *length*, *format*)
  Returns `None`. This function corresponds with `XtSetSelectionParameters`.

`SetSensitive`(*sensitive*)
  Returns `None`. This function corresponds with `XtSetSensitive`.

`SetValues`(*argdict*)
  Returns `None`. This function corresponds with `XtSetValues`.

`SetWMNormalHints`(*hintdict*)
  Returns `None`. This function corresponds with `XSetWMNormalHints`.

`SetWindowColormap`(*cm_object*)
  Returns `None`. This function corresponds with `XSetWindowColormap`.

`Superclass()`
: Returns `widget_class`. This function corresponds with `XtSuperclass`.

`TranslateCoords(`*x*`,` *y*`)`
: Returns `(rootx, rooty)`. This function corresponds with `XtTranslateCoords`.

`UndefineCursor()`
: Returns `None`. This function corresponds with `XUndefineCursor`.

`UngrabButton(`*button*`,` *modifiers*`)`
: Returns `None`. This function corresponds with `XUngrabButton`.

`UninstallTranslations()`
: Returns `None`. This function corresponds with `XtUninstallTranslations`.

`UnmanageChild()`
: Returns `None`. This function corresponds with `XtUnmanageChild`.

`UnmapWidget()`
: Returns `None`. This function corresponds with `XtUnmapWidget`.

`UnrealizeWidget()`
: Returns `None`. This function corresponds with `XtUnrealizeWidget`.

`WidthMMOfScreen()`
: Returns `width_mm`. This function corresponds with `WidthMMOfScreen`.

`WidthOfScreen()`
: Returns `width`. This function corresponds with `WidthOfScreen`.

`Window()`
: Returns `window`. This function corresponds with `XtWindow`.

4. Visual Info object

Visual Info objects are used to represent a XVisualInfo structure. The widget method `GetVisualInfo` returns a list of Visual Info objects; the widget method `DefaultVisualOfScreen` returns a single Visual Info object. Visual Info objects can be used to create a colormap using the widget method `CreateColormap` and to create an Image object using the Visual Info method `CreateImage`.

Visual Info objects (vi_objects) have the following method and variables:

`bits_per_rgb`
: Field in XVisualInfo structure.

`blue_mask`
: Field in XVisualInfo structure.

`c_class`
: Field in XVisualInfo structure. Alternative name for the `class` field.

`class`
: Field in XVisualInfo structure.

`colormap_size`
: Field in XVisualInfo structure.

`depth`
: Field in XVisualInfo structure.

`green_mask`
: Field in XVisualInfo structure.

`red_mask`

Field in XVisualInfo structure.

`visual`

Field in XVisualInfo structure.

`visualid`

Field in XVisualInfo structure.

`CreateColormap(`*alloc*`)`

Returns `cm_object`. This function corresponds with `XCreateColormap`.

`CreateImage(`*depth*`,` *format*`,` *offset*`,` *data*`,` *width*`,` *height*`,` *bitmap_pad*`,` *bytes_per_line*`)`

Returns `im_object`. This function corresponds with `XCreateImage`.

5. Image object

An Image object is used to represent an XImage structure. An Image object is created using the `CreateImage` method of a Visual Info object. The image can be copied to a window or a pixmap using the Graphics Context method `PutImage`.
Image objects (im_objects) have no methods or variables.

6. Graphics Context object

All Graphics Context objects have methods that perform drawing operations. They are listed here, with the argument types they expect. XPoint is a pair of integers; XRectangle is a tuple of four integers; XSegment is a tuple of four integers; XArc is a tuple of six integers; <type>[] stands for a list of <type> objects.
All GC objects have data attributes representing the individual values. These are gc.line_width, gc.line_style, etc. (The dash attribute is not supported.) These attributes can be assigned to if the GC object is writable.
The Pixmap arguments are Pixmap objects; the Font arguments are Font objects; the Image arguments are Image objects.
Graphics Context objects (gc_objects) have the following methods and variables:

`arc_mode`

Field in XGCValues structure.

`background`

Field in XGCValues structure.

`cap_style`

Field in XGCValues structure.

`clip_mask`

Field in XGCValues structure.

`clip_x_origin`

Field in XGCValues structure.

`clip_y_origin`

Field in XGCValues structure.

`dash_offset`

Field in XGCValues structure.

`file_style`

Field in XGCValues structure.

`fill_rule`

Field in XGCValues structure.

`font`
  Field in XGCValues structure.
`foreground`
  Field in XGCValues structure.
`function`
  Field in XGCValues structure.
`graphics_exposures`
  Field in XGCValues structure.
`join_style`
  Field in XGCValues structure.
`line_style`
  Field in XGCValues structure.
`line_width`
  Field in XGCValues structure.
`plane_mask`
  Field in XGCValues structure.
`stipple`
  Field in XGCValues structure.
`subwindow_mode`
  Field in XGCValues structure.
`tile`
  Field in XGCValues structure.
`ts_x_origin`
  Field in XGCValues structure.
`ts_y_origin`
  Field in XGCValues structure.
`ChangeGC(`*gc_dict*`)`
  Returns `None`. This function corresponds with `XChangeGC`.
`DrawArc(`*x*`, `*y*`, `*width*`, `*height*`, `*angle1*`, `*angle2*`)`
  Returns `None`. This function corresponds with `XDrawArc`.
`DrawArcs(`*XArc[]*`)`
  Returns `None`. This function corresponds with `XDrawArcs`.
`DrawImageString(`*x*`, `*y*`, `*string*`)`
  Returns `None`. This function corresponds with `XDrawImageString`.
`DrawLine(`*x1*`, `*y1*`, `*x2*`, `*y2*`)`
  Returns `None`. This function corresponds with `XDrawLine`.
`DrawLines(`*XPoint[]*`, `*mode*`)`
  Returns `None`. This function corresponds with `XDrawLines`.
`DrawPoint(`*x*`, `*y*`)`
  Returns `None`. This function corresponds with `XDrawPoint`.
`DrawPoints(`*XPoint[]*`, `*mode*`)`
  Returns `None`. This function corresponds with `XDrawPoints`.
`DrawRectangle(`*x*`, `*y*`, `*width*`, `*height*`)`
  Returns `None`. This function corresponds with `XDrawRectangle`.
`DrawRectangles(`*XRectangle[]*`)`

Returns `None`. This function corresponds with `XDrawRectangles`.

`DrawSegments(`*XSegment[]*`)`
Returns `None`. This function corresponds with `XDrawSegments`.

`DrawString(`*x*, *y*, *string*`)`
Returns `None`. This function corresponds with `XDrawString`.

`FillArc(`*x*, *y*, *width*, *height*, *angle1*, *angle2*`)`
Returns `None`. This function corresponds with `XFillArc`.

`FillArcs(`*XArc[]*`)`
Returns `None`. This function corresponds with `XFillArcs`.

`FillPolygon(`*XPoint[]*, *shape*, *mode*`)`
Returns `None`. This function corresponds with `XFillPolygon`.

`FillRectangle(`*x*, *y*, *width*, *height*`)`
Returns `None`. This function corresponds with `XFillRectangle`.

`FillRectangles(`*XRectangle[]*`)`
Returns `None`. This function corresponds with `XFillRectangles`.

`PutImage(`*Image*, *src_x*, *src_y*, *dest_x*, *dest_y*, *width*, *height*`)`
Returns `None`. This function corresponds with `XPutImage`.

`SetArcMode(`*arc_mode*`)`
Returns `None`. This function corresponds with `XSetArcMode`.

`SetBackground(`*background*`)`
Returns `None`. This function corresponds with `XSetBackground`.

`SetClipMask(`*Pixmap*`)`
Returns `None`. This function corresponds with `XSetClipMask`.

`SetClipOrigin(`*clip_x_origin*, *clip_y_origin*`)`
Returns `None`. This function corresponds with `XSetClipOrigin`.

`SetClipRectangles(`*clip_x_origin*, *clip_y_origin*, *XRectangle[]*, *ordering*`)`
Returns `None`. This function corresponds with `XSetClipRectangles`.

`SetDashes(`*dash_offset*, *dash_list*`)`
Returns `None`. This function corresponds with `XSetDashes`.

`SetFillRule(`*fill_rule*`)`
Returns `None`. This function corresponds with `XSetFillRule`.

`SetFillStyle(`*fill_style*`)`
Returns `None`. This function corresponds with `XSetFillStyle`.

`SetFont(`*font_object*`)`
Returns `None`. This function corresponds with `XSetFont`.

`SetForeground(`*foreground*`)`
Returns `None`. This function corresponds with `XSetForeground`.

`SetFunction(`*function*`)`
Returns `None`. This function corresponds with `XSetFunction`.

`SetGraphicsExposures(`*graphics_exposure*`)`
Returns `None`. This function corresponds with `XSetGraphicsExposures`.

`SetLineAttributes(`*line_width*, *line_style*, *cap_style*, *join_style*`)`
Returns `None`. This function corresponds with `XSetLineAttributes`.

`SetPlaneMask(`*plane_mask*`)`
Returns `None`. This function corresponds with `XSetPlaneMask`.

`SetRegion`(*region*)
: Returns `None`. This function corresponds with `XSetRegion`.

`SetState`(*foreground*, *background*, *function*, *plane_mask*)
: Returns `None`. This function corresponds with `XSetState`.

`SetStipple`(*stipple*)
: Returns `None`. This function corresponds with `XSetStipple`.

`SetSubwindowMode`(*subwindow_mode*)
: Returns `None`. This function corresponds with `XSetSubwindowMode`.

`SetTSOrigin`(*ts_x_origin*, *ts_y_origin*)
: Returns `None`. This function corresponds with `XSetTSOrigin`.

`SetTile`(*tile*)
: Returns `None`. This function corresponds with `XSetTile`.

7. Colormap object

Colormap objects are used to represent the `Colormap` type. A Colormap object can be obtained using the widget method `CreateColormap` (which also needs a Visual Info object) or the widget method `DefaultColormapOfScreen`. A Colormap object can be used in the widget method `SetWindowColormap` and as the `colormap` resource of widgets.

Colormap objects (cm_objects) have the following methods:

`AllocColor`(*red*, *green*, *blue*)
: Returns `(pixel, red, green, blue, flags)`. This function corresponds with `XAllocColor`.

`AllocColorCells`(*contig*, *nplanes*, *npixels*)
: Returns `(plane_masks_list, pixels_list)`. This function corresponds with `XAllocColorCells`.

`AllocNamedColor`(*color_name*)
: Returns `((screen_def.pixel, screen_def.red, screen_def.green, screen_def.blue, screen_def.flags), (exact_def.pixel, exact_def.red, exact_def.green, exact_def.blue, exact_def.flags))`. This function corresponds with `XAllocNamedColor`.

`FreeColors`(*pixel_list*, *planes*)
: Returns `None`. This function corresponds with `XFreeColors`.

`Install`()
: Returns `None`. This function corresponds with `XInstallColormap`.

`InstallColormap`()
: Returns `None`. This function corresponds with `XInstallColormap`.

`LookupColor`(*color_name*)
: Returns `((exact_def.pixel, exact_def.red, exact_def.green, exact_def.blue, exact_def.flags), (screen_def.pixel, screen_def.red, screen_def.green, screen_def.blue, screen_def.flags))`. This function corresponds with `XLookupColor`.

`ParseColor`(*color_name*)
: Returns `(pixel, red, green, blue, flags)`. This function corresponds with `XParseColor`.

`QueryColor`(*pixel*)
: Returns `(pixel, red, green, blue, flags)`. This function corresponds with `XQueryColor`.

`QueryColors(`*pixel_list*`)`
Returns `XColor_list`. This function corresponds with `XQueryColors`.
`StoreColors(`*color_list*`)`
Returns `None`. This function corresponds with `XStoreColors`.

8. Pixmap object

Pixmap objects are used to represent the `Pixmap` type. A Pixmap object can be obtained using the widget method `CreatePixmap`.
A Pixmap is an offscreen piece of memory in which the application can draw. Drawing in the pixmap can be done using a Graphics Context object which can be created using the `CreateGC` method. The pixmap can be copied to a window or another pixmap using the `CopyArea` and `CopyPlane` methods.
Pixmap objects (pixmap_objects) have the following methods:
`CopyArea(`*destobject*`,` *gc_object*`,` *src_x*`,` *src_y*`,` *width*`,` *height*`,` *dest_x*`,` *dest_y*`)`
Returns `None`. This function corresponds with `XCopyArea`.
`CopyPlane(`*destobject*`,` *gc_object*`,` *src_x*`,` *src_y*`,` *width*`,` *height*`,` *dest_x*`,` *dest_y*`,` *plane*`)`
Returns `None`. This function corresponds with `XCopyPlane`.
`CreateGC(`*gc_dict*`)`
Returns `gc_object`. This function corresponds with `XCreateGC`.
`ReadBitmapFile(`*file*`)`
Returns `(width, height, pixmap_object, hotspot_x, hotspot_y)`. This function corresponds with `XReadBitmapFile`.
`WriteBitmapFile(`*file*`,` *width*`,` *height* [`,` *x_hot*`,` *y_hot*] `)`
Returns `None`. This function corresponds with `XWriteBitmapFile`.

9. Font object

The Font object attributes are read-only. The Font ID (fid) is fairly useless. Everywhere a Font ID is required in the library, a Font object must be given.
Font objects (font_objects) have the following methods and variables:
`all_chars_exist`
Field in XFontStruct structure.
`ascent`
Field in XFontStruct structure.
`default_char`
Field in XFontStruct structure.
`descent`
Field in XFontStruct structure.
`direction`
Field in XFontStruct structure.
`fid`
Field in XFontStruct structure.
`max_bounds`
Field in XFontStruct structure.
`max_byte1`
Field in XFontStruct structure.

`max_char_or_byte2`
: Field in XFontStruct structure.

`min_bounds`
: Field in XFontStruct structure.

`min_byte1`
: Field in XFontStruct structure.

`min_char_or_byte2`
: Field in XFontStruct structure.

`n_properties`
: Field in XFontStruct structure.

`per_char`
: Field in XFontStruct structure.

`properties`
: Field in XFontStruct structure.

`TextExtents(`*string*`)`
: Returns `(direction, font_ascent, font_descent, (lbearing, rbearing, width, ascent, descent, attributes))`. This function corresponds with `XTextExtents`.

`TextWidth(`*string*`)`
: Returns `width`. This function corresponds with `XTextWidth`.

10. Translation object

Translation objects are used to represent the `XtTranslations` type. A Translation object can be obtained using the `Xt` function `ParseTranslationTable`. Translation objects do not have any methods or variables.

11. Callback object

Callback options are used to represent the system-generated data that is passed to callback functions. Callback options do no have any methods. The data attributes depend on the type of callback object. Callback objects are created by the system to represent the call_data argument of callback functions. Callback objects that represent XEvent structures can be created using the `Xlib` function `Event`.

Chapter 4
Built-in module Xrm

1. Xrm functions and variables

The module defines the following functions:

`GetFileDatabase(`*filename*`)`

Returns `database_object`. This function corresponds with `XrmGetFileDatabase`.

`GetStringDatabase(`*string*`)`

Returns `database_object`. This function corresponds with `XrmGetStringDatabase`.

2. Xrm Widget object

After Xrm has been imported at least once somewhere in the program, all widget objects hage the following extra method.

`ScreenDatabase()`

Returns `database_object`. This function corresponds with `XtScreenDatabase`.

3. Xrm Display object

After Xrm has been imported at least once somewhere in the program, display objects have the following extra methods.

`GetDatabase()`

Returns `database_object`. This function corresponds with `XrmGetDatabase`.

`SetDatabase(`*database*`)`

Returns `None`. This function corresponds with `XrmSetDatabase`.

4. Database object

Database objects are used to represent the `XrmDatabase` type. A database object can be obtained using the widget method `ScreenDatabase`, the display method `GetDatabase`, and the `Xrm` functions `GetFileDatabase` and `GetStringDatabase`.

`DestroyDatabase()`

Returns `None`. This function corresponds with `DestroyDatabase`.

`GetResource`(*name*, *class*)
: Returns `None` or `(type, value)`. This function corresponds with `XrmGetResource`.

`LocaleOfDatabase()`
: Returns `locale`. This function corresponds with `XrmLocaleOfDatabase`.

`PutFileDatabase`(*filename*)
: Returns `None`. This function corresponds with `XrmPutFileDatabase`.

`PutLineResource`(*line*)
: Returns `None`. This function corresponds with `XrmPutLineResource`.

`PutResource`(*specifier*, *type*, *value*)
: Returns `None`. This function corresponds with `XrmPutResource`.

`PutStringResource`(*specifier*, *value*)
: Returns `None`. This function corresponds with `XrmPutStringResource`.

Chapter 5
Built-in module Xm

Which of the variables and methods are available depends on the version of the Motif libraries. For instance, the variable `TextField` is only available in Motif versions 1.1 and higher, the variable `Display` is only available in Motif version 1.2 and higher. To check dynamically which Motif version is being used, one can use the variable `VERSION`, `REVISION`, `UPDATE_LEVEL`, and `Version`.

1. Xm functions and variables

`ArrowButtonGadget`
: Widget class object.

`BulletinBoard`
: Widget class object.

`CascadeButton`
: Widget class object.

`CascadeButtonGadget`
: Widget class object.

`Command`
: Widget class object.

`DialogShell`
: Widget class object.

`Display`
: Widget class object.

`DragContext`
: Widget class object.

`DragIcon`
: Widget class object.

`DrawingArea`
: Widget class object.

`DrawnButton`
Widget class object.

`DropSite`
Widget class object.

`DropTransfer`
Widget class object.

`FileSelectionBox`
Widget class object.

`Form`
Widget class object.

`Frame`
Widget class object.

`Gadget`
Widget class object.

`Label`
Widget class object.

`LabelGadget`
Widget class object.

`List`
Widget class object.

`MainWindow`
Widget class object.

`MenuShell`
Widget class object.

`MessageBox`
Widget class object.

`PanedWindow`
Widget class object.

`PushButton`
Widget class object.

`PushButtonGadget`
Widget class object.

`REVISION`
Motif minor version number.

`RowColumn`
Widget class object.

`Scale`
Widget class object.

`ScrollBar`
Widget class object.

`ScrolledWindow`
Widget class object.

`SelectionBox`
Widget class object.

`Separator`
    Widget class object.
`SeparatorGadget`
    Widget class object.
`Text`
    Widget class object.
`TextField`
    Widget class object.
`ToggleButton`
    Widget class object.
`ToggleButtonGadget`
    Widget class object.
`UPDATE_LEVEL`
    Motif update level.
`VERSION`
    Motif major version number.
`Version`
    Motif major and minor version number.

2. Xm Widget Class object

Motif Widget Class objects have the following method in addition to the standard ones:

`GetSecondaryResourceData()`
    Returns `resource_list_list`. This function corresponds with `XmGetSecondaryResourceData`.

3. Xm Widget object

If Motif is included all widget objects have the following methods in addition to the normal widget methods. These methods are only available after Xm was imported at least once somewhere in the program.

`AddTabGroup()`
    Returns `None`. This function corresponds with `XmAddTabGroup`.
`ChangeColor(`*background*`)`
    Returns `None`. This function corresponds with `XmChangeColor`.
`ConvertUnits(`*orientation*`, `*from_unit_type*`, `*from_value*`, `*to_unit_type*`)`
    Returns `None`. This function corresponds with `XmConvertUnits`.
`CreateArrowButton(`*name*`, `*argdict*`)`
    Returns `widget`. This function corresponds with `XmCreateArrowButton`.
`CreateArrowButtonGadget(`*name*`, `*argdict*`)`
    Returns `widget`. This function corresponds with `XmCreateArrowButtonGadget`.
`CreateBulletinBoard(`*name*`, `*argdict*`)`
    Returns `widget`. This function corresponds with `XmCreateBulletinBoard`.
`CreateBulletinBoardDialog(`*name*`, `*argdict*`)`
    Returns `widget`. This function corresponds with `XmCreateBulletinBoardDialog`.
`CreateCascadeButton(`*name*`, `*argdict*`)`
    Returns `widget`. This function corresponds with `XmCreateCascadeButton`.

`CreateCascadeButtonGadget(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateCascadeButtonGadget`.

`CreateCommand(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateCommand`.

`CreateDialogShell(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateDialogShell`.

`CreateDragIcon(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateDragIcon`.

`CreateDrawingArea(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateDrawingArea`.

`CreateDrawnButton(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateDrawnButton`.

`CreateErrorDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateErrorDialog`.

`CreateFileSelectionBox(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateFileSelectionBox`.

`CreateFileSelectionDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateFileSelectionDialog`.

`CreateForm(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateForm`.

`CreateFormDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateFormDialog`.

`CreateFrame(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateFrame`.

`CreateInformationDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateInformationDialog`.

`CreateLabel(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateLabel`.

`CreateLabelGadget(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateLabelGadget`.

`CreateList(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateList`.

`CreateMainWindow(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateMainWindow`.

`CreateMenuBar(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateMenuBar`.

`CreateMenuShell(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateMenuShell`.

`CreateMessageBox(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateMessageBox`.

`CreateMessageDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateMessageDialog`.

`CreateOptionMenu(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateOptionMenu`.

`CreatePanedWindow(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreatePanedWindow`.

`CreatePopupMenu(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreatePopupMenu`.

`CreatePromptDialog(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreatePromptDialog`.

`CreatePulldownMenu(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreatePulldownMenu`.

`CreatePushButton(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreatePushButton`.

`CreatePushButtonGadget(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreatePushButtonGadget`.

`CreateQuestionDialog(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateQuestionDialog`.

`CreateRadioBox(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateRadioBox`.

`CreateRowColumn(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateRowColumn`.

`CreateScale(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateScale`.

`CreateScrollBar(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateScrollBar`.

`CreateScrolledList(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateScrolledList`.

`CreateScrolledText(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateScrolledText`.

`CreateScrolledWindow(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateScrolledWindow`.

`CreateSelectionBox(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSelectionBox`.

`CreateSelectionDialog(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSelectionDialog`.

`CreateSeparator(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSeparator`.

`CreateSeparatorGadget(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSeparatorGadget`.

`CreateSimpleCheckBox(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSimpleCheckBox`.

`CreateSimpleMenuBar(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSimpleMenuBar`.

`CreateSimpleOptionMenu(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSimpleOptionMenu`.

`CreateSimplePopupMenu(`*name*`,` *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSimplePopupMenu`.

`CreateSimplePulldownMenu(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSimplePulldownMenu`.

`CreateSimpleRadioBox(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateSimpleRadioBox`.

`CreateTemplateDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateTemplateDialog`.

`CreateText(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateText`.

`CreateTextField(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateTextField`.

`CreateToggleButton(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateToggleButton`.

`CreateToggleButtonGadget(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateToggleButtonGadget`.

`CreateWarningDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateWarningDialog`.

`CreateWorkArea(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateWorkArea`.

`CreateWorkingDialog(`*name*, *argdict*`)`
Returns `widget`. This function corresponds with `XmCreateWorkingDialog`.

`DragStart(`*event*, *argdict*`)`
Returns `dragcontext_widget`. This function corresponds with `XmDragStart`.

`DropSiteEndUpdate()`
Returns `None`. This function corresponds with `XmDropSiteEndUpdate`.

`DropSiteRegister(`*argdict*`)`
Returns `None`. This function corresponds with `XmDropSiteRegister`.

`DropSiteStartUpdate()`
Returns `None`. This function corresponds with `XmDropSiteStartUpdate`.

`DropSiteUnregister()`
Returns `None`. This function corresponds with `XmDropSiteUnregister`.

`DropSiteUpdate(`*argdict*`)`
Returns `None`. This function corresponds with `XmDropSiteUpdate`.

`GetDisplay()`
Returns `display_widget`. This function corresponds with `XmGetDisplay`.

`GetDragContext(`*timestamp*`)`
Returns `dragcontext_widget`. This function corresponds with `XmGetDragContext`.

`GetFocusWidget()`
Returns `widget`. This function corresponds with `XmGetFocusWidget`.

`GetTabGroup()`
Returns `widget`. This function corresponds with `XmGetTabGroup`.

`GetVisibility()`
Returns `visibility`. This function corresponds with `XmGetVisibility`.

`GetXmScreen()`
Returns `widget`. This function corresponds with `XmGetXmScreen`.

`IsTraversable()`
    Returns `success`. This function corresponds with `XmIsTraversable`.
`ProcessTraversal(`*direction*`)`
    Returns `success`. This function corresponds with `XmProcessTraversal`.
`RemoveTabGroup()`
    Returns `None`. This function corresponds with `XmRemoveTabGroup`.
`TrackingEvent(`*cursor*`,` *confine_to*`)`
    Returns `(widget, event)`. This function corresponds with `XmTrackingEvent`.
`TrackingLocate(`*cursor*`,` *confine_to*`)`
    Returns `widget`. This function corresponds with `XmTrackingLocate`.
`UpdateDisplay()`
    Returns `None`. This function corresponds with `XmUpdateDisplay`.

3.1 Shell widget methods

Widget objects whose widget class is subclassed from Shell have the following additional method:
`IsMotifWMRunning()`
    Returns `success`. This function corresponds with `XmIsMotifWMRunning`.

3.2 VendorShell widget methods

Widget objects whose widget class is subclassed from VendorShell have the following additional methods (note that VendorShell is a subclass of Shell):
`ActivateProtocol(`*property*`,` *protocol*`)`
    Returns `None`. This function corresponds with `XmActivateProtocol`.
`ActivateWMProtocol(`*protocol*`)`
    Returns `None`. This function corresponds with `XmActivateWMProtocol`.
`AddProtocolCallback(`*property*`,` *protocol*`,` *cbfunc*`,` *cbarg*`)`
    Returns `None`. This function corresponds with `XmAddProtocolCallback`.
`AddProtocols(`*property*`,` *protocol[]*`)`
    Returns `None`. This function corresponds with `XmAddProtocols`.
`AddWMProtocolCallback(`*protocol*`,` *cbfunc*`,` *cbarg*`)`
    Returns `None`. This function corresponds with `XmAddWMProtocolCallback`.
`AddWMProtocols(`*protocol[]*`)`
    Returns `None`. This function corresponds with `XmAddWMProtocols`.
`DeactivateProtocol(`*property*`,` *protocol*`)`
    Returns `None`. This function corresponds with `XmDeactivateProtocol`.
`DeactivateWMProtocol(`*protocol*`)`
    Returns `None`. This function corresponds with `XmDeactivateWMProtocol`.
`RemoveProtocolCallback(`*property*`,` *protocol*`,` *callback*`,` *closure*`)`
    Returns `None`. This function corresponds with `XmRemoveProtocolCallback`.
`RemoveProtocols(`*property*`,` *protocol[]*`)`
    Returns `None`. This function corresponds with `XmRemoveProtocols`.
`RemoveWMProtocolCallback(`*protocol*`,` *callback*`,` *closure*`)`
    Returns `None`. This function corresponds with `XmRemoveWMProtocolCallback`.

`RemoveWMProtocols(`*protocol[]*`)`
  Returns `None`. This function corresponds with `XmRemoveWMProtocols`.
`SetProtocolHooks(`*property*`,` *protocol*`,` *prehook*`,` *pre_closure*`,` *posthook*`,` *post_closure*`)`
  Returns `None`. This function corresponds with `XmSetProtocolHooks`.
`SetWMProtocolHooks(`*protocol*`,` *prehook*`,` *pre_closure*`,` *posthook*`,` *post_closure*`)`
  Returns `None`. This function corresponds with `XmSetWMProtocolHooks`.

3.3 XmCascadeButton widget methods

Widget objects whose widget class is subclassed from XmCascadeButton or XmCascadeButtonGadget have the following additional method:

`CascadeButtonHighlight(`*highlight*`)`
  Returns `None`. This function corresponds with `XmCascadeButtonHighlight`.

3.4 XmCommand widget methods

Widget objects whose widget class is subclassed from XmCommand have the following additional methods:

`CommandAppendValue(`*command*`)`
  Returns `None`. This function corresponds with `XmCommandAppendValue`.
`CommandError(`*error*`)`
  Returns `None`. This function corresponds with `XmCommandError`.
`CommandGetChild(`*child*`)`
  Returns `child_widget`. This function corresponds with `XmCommandGetChild`.
`CommandSetValue(`*command*`)`
  Returns `None`. This function corresponds with `XmCommandSetValue`.

3.5 XmDragContext widget methods

Widget objects whose widget class is subclassed from XmDragContext have the following additional methods:

`DragCancel()`
  Returns `None`. This function corresponds with `XmDragCancel`.
`DropTransferStart(`*argdict*`)`
  Returns `droptransfer_widget`. This function corresponds with `XmDropTransferStart`.

3.6 XmDropSite widget methods

Widget objects whose widget class is subclassed from XmDropSite have the following additional methods:

`DropSiteConfigureStackingOrder(`*sibling*`,` *stack_mode*`)`
  Returns `None`. This function corresponds with `XmDropSiteConfigureStackingOrder`.

3.7 XmFileSelectionBox widget methods

Widget objects whose widget class is subclassed from XmFileSelectionBox have the following additional methods (note that XmFileSelectionBox is a subclass of XmSelectionBox):

`FileSelectionBoxGetChild(`*child*`)`
  Returns `child_widget`. This function corresponds with `XmFileSelectionBoxGetChild`.

`FileSelectionDoSearch(`*dirmask*`)`
Returns `None`. This function corresponds with `XmFileSelectionDoSearch`.

3.8 XmList widget methods

Widget objects whose widget class is subclassed from XmList have the following additional methods:

`ListAddItem(`*item*`, `*position*`)`
Returns `None`. This function corresponds with `XmListAddItem`.

`ListAddItemUnselected(`*item*`, `*position*`)`
Returns `None`. This function corresponds with `XmListAddItemUnselected`.

`ListAddItems(`*item_list*`, `*position*`)`
Returns `None`. This function corresponds with `XmListAddItems`.

`ListAddItemsUnselected(`*item_list*`, `*position*`)`
Returns `None`. This function corresponds with `XmListAddItemsUnselected`.

`ListDeleteAllItems()`
Returns `None`. This function corresponds with `XmListDeleteAllItems`.

`ListDeleteItem(`*item*`)`
Returns `None`. This function corresponds with `XmListDeleteItem`.

`ListDeleteItems(`*item_list*`)`
Returns `None`. This function corresponds with `XmListDeleteItems`.

`ListDeleteItemsPos(`*item_count*`, `*position*`)`
Returns `None`. This function corresponds with `XmListDeleteItemsPos`.

`ListDeletePos(`*position*`)`
Returns `None`. This function corresponds with `XmListDeletePos`.

`ListDeletePositions(`*positionlist*`)`
Returns `None`. This function corresponds with `XmListDeletePositions`.

`ListDeselectAllItems()`
Returns `None`. This function corresponds with `XmListDeselectAllItems`.

`ListDeselectItem(`*item*`)`
Returns `None`. This function corresponds with `XmListDeselectItem`.

`ListDeselectPos(`*position*`)`
Returns `None`. This function corresponds with `XmListDeselectPos`.

`ListGetKbdItemPos()`
Returns `position`. This function corresponds with `XmListGetKbdItemPos`.

`ListGetSelectedPos()`
Returns `position_list` or `None`. This function corresponds with `XmListGetSelectedPos`.

`ListItemExists(`*item*`)`
Returns `success`. This function corresponds with `XmListItemExists`.

`ListItemPos(`*item*`)`
Returns `position`. This function corresponds with `XmListItemPos`.

`ListPosSelected(`*position*`)`
Returns `selected`. This function corresponds with `XmListPosSelected`.

`ListPosToBounds(`*position*`)`
Returns `(x, y, width, height)` or `None`. This function corresponds with

`XmListPosToBounds`.

`ListReplaceItems(`*old_items_list*`,` *new_items_list*`)`
Returns `None`. This function corresponds with `XmListReplaceItems`.

`ListReplaceItemsPos(`*new_item_list*`,` *position*`)`
Returns `None`. This function corresponds with `XmListReplaceItemsPos`.

`ListReplaceItemsPosUnselected(`*new_item_list*`,` *position*`)`
Returns `None`. This function corresponds with `XmListReplaceItemsPosUnselected`.

`ListReplaceItemsUnselected(`*old_items_list*`,` *new_items_list*`)`
Returns `None`. This function corresponds with `XmListReplaceItemsUnselected`.

`ListReplacePositions(`*position_list*`,` *item_list*`)`
Returns `None`. This function corresponds with `XmListReplacePositions`.

`ListSelectItem(`*item*`,` *notify*`)`
Returns `None`. This function corresponds with `XmListSelectItem`.

`ListSelectPos(`*position*`,` *notify*`)`
Returns `None`. This function corresponds with `XmListSelectPos`.

`ListSetAddMode(`*state*`)`
Returns `None`. This function corresponds with `XmListSetAddMode`.

`ListSetBottomItem(`*item*`)`
Returns `None`. This function corresponds with `XmListSetBottomItem`.

`ListSetBottomPos(`*position*`)`
Returns `None`. This function corresponds with `XmListSetBottomPos`.

`ListSetHorizPos(`*position*`)`
Returns `None`. This function corresponds with `XmListSetHorizPos`.

`ListSetItem(`*item*`)`
Returns `None`. This function corresponds with `XmListSetItem`.

`ListSetKbdItemPos(`*position*`)`
Returns `success`. This function corresponds with `XmListSetKbdItemPos`.

`ListSetPos(`*position*`)`
Returns `None`. This function corresponds with `XmListSetPos`.

`ListUpdateSelectedList()`
Returns `None`. This function corresponds with `XmListUpdateSelectedList`.

`ListYToPos(`*y*`)`
Returns `position`. This function corresponds with `XmListYToPos`.

3.9 XmMainWindow widget methods

Widget objects whose widget class is subclassed from XmMainWindow have the following additional methods:

`MainWindowSep1()`
Returns `widget`. This function corresponds with `XmMainWindowSep1`.

`MainWindowSep2()`
Returns `widget`. This function corresponds with `XmMainWindowSep2`.

`MainWindowSep3()`
Returns `widget`. This function corresponds with `XmMainWindowSep3`.

`MainWindowSetAreas(`*menu_bar*`,` *command_window*`,` *horizontal_scrollbar*`,` *vertical_scrollbar*`,` *work_region*`)`
Returns `None`. This function corresponds with `XmMainWindowSetAreas`.

3.10 XmMessageBox widget methods

Widget objects whose widget class is subclassed from XmMessageBox have the following additional method:

`MessageBoxGetChild(`*child*`)`
: Returns `child_widget`. This function corresponds with `XmMessageBoxGetChild`.

3.11 XmRowColumn widget methods

Widget objects whose widget class is subclassed from XmRowColumn have the following additional methods:

`GetPostedFromWidget()`
: Returns `widget`. This function corresponds with `XmGetPostedFromWidget`.

`GetTearOffControl()`
: Returns `widget`. This function corresponds with `XmGetTearOffControl`.

`MenuPosition(`*event*`)`
: Returns `None`. This function corresponds with `XmMenuPosition`.

`OptionButtonGadget()`
: Returns `widget`. This function corresponds with `XmOptionButtonGadget`.

`OptionLabelGadget()`
: Returns `widget`. This function corresponds with `XmOptionLabelGadget`.

3.12 XmScale widget methods

Widget objects whose widget class is subclassed from XmScale have the following additional methods:

`ScaleGetValue()`
: Returns `value`. This function corresponds with `XmScaleGetValue`.

`ScaleSetValue(`*value*`)`
: Returns `None`. This function corresponds with `XmScaleSetValue`.

3.13 XmScrollBar widget methods

Widget objects whose widget class is subclassed from XmScrollBar have the following additional methods:

`ScrollBarGetValues()`
: Returns `(value, slider_size, increment, page_increment)`. This function corresponds with `XmScrollBarGetValues`.

`ScrollBarSetValues(`*value*`,` *slider_size*`,` *increment*`,` *page_increment*`,` *notify*`)`
: Returns `None`. This function corresponds with `XmScrollBarSetValues`.

3.14 XmScrolledWindow widget methods

Widget objects whose widget class is subclassed from XmScrolledWindow have the following additional methods:

`ScrollVisible(`*widget*`,` *left_right_margin*`,` *top_bottom_margin*`)`
: Returns `None`. This function corresponds with `XmScrollVisible`.

`ScrolledWindowSetAreas(`*horizontal_scrollbar*`,` *vertical_scrollbar*`,` *work_region*`)`
: Returns `None`. This function corresponds with `XmScrolledWindowSetAreas`.

3.15 XmSelectionBox widget methods

Widget objects whose widget class is subclassed from XmSelectionBox have the following additional method:

`SelectionBoxGetChild(`*child*`)`
: Returns `child_widget`. This function corresponds with `XmSelectionBoxGetChild`.

3.16 XmText widget methods

Widget objects whose widget class is subclassed from XmText have the following additional methods:

`TextClearSelection(`*time*`)`
: Returns `None`. This function corresponds with `XmTextClearSelection`.

`TextCopy(`*time*`)`
: Returns `None`. This function corresponds with `XmTextCopy`.

`TextCut(`*time*`)`
: Returns `None`. This function corresponds with `XmTextCut`.

`TextDisableRedisplay()`
: Returns `None`. This function corresponds with `XmTextDisableRedisplay`.

`TextEnableRedisplay()`
: Returns `None`. This function corresponds with `XmTextEnableRedisplay`.

`TextFindString(`*start*`, `*string*`, `*direction*`)`
: Returns `position` or `None`. This function corresponds with `XmTextFindString`.

`TextGetBaseline()`
: Returns `basline`. This function corresponds with `XmTextGetBaseline`.

`TextGetEditable()`
: Returns `state`. This function corresponds with `XmTextGetEditable`.

`TextGetInsertionPosition()`
: Returns `position`. This function corresponds with `XmTextGetInsertionPosition`.

`TextGetLastPosition()`
: Returns `position`. This function corresponds with `XmTextGetLastPosition`.

`TextGetMaxLength()`
: Returns `length`. This function corresponds with `XmTextGetMaxLength`.

`TextGetSelection()`
: Returns `text`. This function corresponds with `XmTextGetSelection`.

`TextGetSelectionPosition()`
: Returns `(owner, left, right)`. This function corresponds with `XmTextGetSelectionPosition`.

`TextGetSource()`
: Returns `source`. This function corresponds with `XmTextGetSource`.

`TextGetString()`
: Returns `text`. This function corresponds with `XmTextGetString`.

`TextGetTopCharacter()`
: Returns `position`. This function corresponds with `XmTextGetTopCharacter`.

`TextInsert(`*position*`, `*value*`)`
: Returns `None`. This function corresponds with `XmTextInsert`.

`TextPaste()`
Returns `is_owner`. This function corresponds with `XmTextPaste`.
`TextPosToXY(`*position*`)`
Returns `(x, y)` or `None`. This function corresponds with `XmTextPosToXY`.
`TextRemove()`
Returns `success`. This function corresponds with `XmTextRemove`.
`TextReplace(`*from_pos*`,` *to_pos*`,` *value*`)`
Returns `None`. This function corresponds with `XmTextReplace`.
`TextScroll(`*lines*`)`
Returns `None`. This function corresponds with `XmTextScroll`.
`TextSetAddMode(`*state*`)`
Returns `None`. This function corresponds with `XmTextSetAddMode`.
`TextSetEditable(`*editable*`)`
Returns `None`. This function corresponds with `XmTextSetEditable`.
`TextSetHighlight(`*left*`,` *right*`,` *mode*`)`
Returns `None`. This function corresponds with `XmTextSetHighlight`.
`TextSetInsertionPosition(`*position*`)`
Returns `None`. This function corresponds with `XmTextSetInsertionPosition`.
`TextSetMaxLength(`*max_length*`)`
Returns `None`. This function corresponds with `XmTextSetMaxLength`.
`TextSetSelection(`*first*`,` *last*`,` *time*`)`
Returns `None`. This function corresponds with `XmTextSetSelection`.
`TextSetSource(`*source*`,` *top_character*`,` *cursor_position*`)`
Returns `None`. This function corresponds with `XmTextSetSource`.
`TextSetString(`*value*`)`
Returns `None`. This function corresponds with `XmTextSetString`.
`TextSetTopCharacter(`*top_character*`)`
Returns `None`. This function corresponds with `XmTextSetTopCharacter`.
`TextShowPosition(`*position*`)`
Returns `None`. This function corresponds with `XmTextShowPosition`.
`TextXYToPos(`x`,` y`)`
Returns `None`. This function corresponds with `XmTextXYToPos`.

3.17 XmTextField widget methods

Widget objects whose widget class is subclassed from XmTextField have the following additional methods:

`TextFieldClearSelection(`*time*`)`
Returns `None`. This function corresponds with `XmTextFieldClearSelection`.
`TextFieldCopy(`*time*`)`
Returns `None`. This function corresponds with `XmTextFieldCopy`.
`TextFieldCut(`*time*`)`
Returns `None`. This function corresponds with `XmTextFieldCut`.
`TextFieldGetBaseline()`
Returns `baseline`. This function corresponds with `XmTextFieldGetBaseline`.

`TextFieldGetEditable()`
Returns `state`. This function corresponds with `XmTextFieldGetEditable`.

`TextFieldGetInsertionPosition()`
Returns `position`. This function corresponds with `XmTextFieldGetInsertionPosition`.

`TextFieldGetLastPosition()`
Returns `position`. This function corresponds with `XmTextFieldGetLastPosition`.

`TextFieldGetMaxLength()`
Returns `length`. This function corresponds with `XmTextFieldGetMaxLength`.

`TextFieldGetSelection()`
Returns `text`. This function corresponds with `XmTextFieldGetSelection`.

`TextFieldGetSelectionPosition()`
Returns `(owner, left, right)`. This function corresponds with `XmTextFieldGetSelectionPosition`.

`TextFieldGetString()`
Returns `text`. This function corresponds with `XmTextFieldGetString`.

`TextFieldInsert(`*position*`,` *value*`)`
Returns `None`. This function corresponds with `XmTextFieldInsert`.

`TextFieldPaste()`
Returns `is_owner`. This function corresponds with `XmTextFieldPaste`.

`TextFieldPosToXY(`*position*`)`
Returns `(x, y)` or `None`. This function corresponds with `XmTextFieldPosToXY`.

`TextFieldRemove()`
Returns `success`. This function corresponds with `XmTextFieldRemove`.

`TextFieldReplace(`*from_pos*`,` *to_pos*`,` *value*`)`
Returns `None`. This function corresponds with `XmTextFieldReplace`.

`TextFieldSetAddMode(`*state*`)`
Returns `None`. This function corresponds with `XmTextFieldSetAddMode`.

`TextFieldSetEditable(`*editable*`)`
Returns `None`. This function corresponds with `XmTextFieldSetEditable`.

`TextFieldSetHighlight(`*left*`,` *right*`,` *mode*`)`
Returns `None`. This function corresponds with `XmTextFieldSetHighlight`.

`TextFieldSetInsertionPosition(`*position*`)`
Returns `None`. This function corresponds with `XmTextFieldSetInsertionPosition`.

`TextFieldSetMaxLength(`*max_length*`)`
Returns `None`. This function corresponds with `XmTextFieldSetMaxLength`.

`TextFieldSetSelection(`*first*`,` *last*`,` *time*`)`
Returns `None`. This function corresponds with `XmTextFieldSetSelection`.

`TextFieldSetString(`*value*`)`
Returns `None`. This function corresponds with `XmTextFieldSetString`.

`TextFieldShowPosition(`*position*`)`
Returns `None`. This function corresponds with `XmTextFieldShowPosition`.

`TextFieldXYToPos(`*x*`,` *y*`)`
Returns `None`. This function corresponds with `XmTextFieldXYToPos`.

3.18 XmToggleButton widget methods

Widget objects whose widget class is subclassed from XmToggleButton have the following additional methods:

`ToggleButtonGetState()`
: Returns `state`. This function corresponds with `XmToggleButtonGetState`.

`ToggleButtonSetState(`*state*`,` *notify*`)`
: Returns `None`. This function corresponds with `XmToggleButtonSetState`.

3.19 XmToggleButtonGadget widget methods

Widget objects whose widget class is subclassed from XmToggleButtonGadget have the following additional methods:

`ToggleButtonGadgetGetState()`
: Returns `state`. This function corresponds with `XmToggleButtonGadgetGetState`.

`ToggleButtonGadgetSetState(`*state*`,` *notify*`)`
: Returns `None`. This function corresponds with `XmToggleButtonGadgetSetState`.

4. Xm Display object

If Motif is included, display object instances have the following extra method.

`GetXmDisplay()`
: Returns `widget`. This function corresponds with `XmGetXmDisplay`.

Chapter 6
Built-in module Mrm

1. Mrm functions and variables

Written by Bob Novas, Century Computing, Inc., updated and converted to LaTeX by Sjoerd Mullender.

The Motif Resource Manager (Mrm) and the User Interface Language (UIL) provide a convenient, standard interface to many commercial GUI-builders. UIL describes the static initial state of a user interface for a widget based application. A UIL module describes a widget hierarchy, the resources of the widgets in the hierarchy, and the callbacks of those widgets. Mrm provides programmatic access to UIL-module definitions.

One powerful feature provided by this Mrm binding is the ability for an interface designer to create an interface using a GUI-builder, bind the UIL-defined interface to (existing) Python functions, and produce a functional GUI program without having to write code (or, at least, not much). We believe that this capability can be used to quickly prototype applications; and, that use of these prototypes operationally is not out of the question.

The Mrm binding is in module Mrm. As usual, you must `import Mrm` to use the Mrm functions.

The Mrm module defines the following function:

`OpenHierarchy`(*filename_or_filename_list*)

Returns `mrm_hierarchy_object`. This function corresponds with `MrmOpenHierarchy`.

This Python function allocates a Mrm ID and opens all the UID files named. A single file name or a list of file names may be passed. This function returns an mrmhierarchyobject that can be used to fetch widget trees from the hierarchy. The function also registers the function `PyEval(string)` (see below) with Mrm for the hierarchy. The function must be called before calling any other Mrm module function but after calling `Xt.Initialize()`.

Examples of using this function are:

```
import Xt, Mrm
```

```
toplev = Xt.Initialize()    # must be called first
mrm_hier1 = Mrm.OpenHierarchy("file-1.uid")
mrm_hier2 = Mrm.OpenHierarchy(["file-1.uid","f2.uid"])
```

A .uid file is the output of the UIL compiler, which is typically invoked:

```
uil file-1.uil -o file-1.uid
```

A Python run-time exception is invoked if the uid file cannot be found.

The `mrm_hierarchy_object` has the following methods:

`FetchWidget(`*index*`,` *parent_widget*`)`

Returns `widget`. This function corresponds with `MrmFetchWidget`.

This Python function fetches the UIL-object named `index` and its children from the mrm_hierarchy object and creates these widgets in a widget-tree under parent_widget. `FetchWidget` returns the UIL-object named `index` wrapped as a Python widget object. Widgets that are children of `index` are created but are not wrapped as Python widget objects. None of the widgets created are managed.

The following Python code snippet shows the use of the FetchWidget method and also shows how to obtain a child widget of `index` as a Python widget object by using the widget `NameToWidget` method.

```
top_level = Xt.Initialize()
mrm_hier = Mrm.OpenHierarchy("panel_cp.uid")
main_w = mrm_hier.FetchWidget("main", top_level)
child_w = main_w.NameToWidget("child-widget-name")
```

`"child-widget-name"` is the partially qualified name of the desired widget child of `main_w`; see XtNameToWidget() for details. Usually the simple widget name can be prefixed with an asterisk to fetch the appropriate widget even if it is not the immediate child of the main widget.

`PyRegister(`*function_object*`,` *argument_type* [`,` *function_name*] `)`

Returns `None`. This function corresponds more-or-less with `MrmRegisterNamesInHierarchy`.

This Python function registers the Python function `function_object` with Mrm so that the Python function `function_object` can be declared as a callback function in UIL. `argument_type` is a string that declares the type of the client argument to `function_object`. The only types currently supported are "integer", "string", and "widget".

`function_object` must be declared in Python with three arguments, which correspond to the standard callback widget, clientdata and calldata arguments. `argument_type` causes the clientdata argument passed to `function_object` to be wrapped as a Python integer, string or widget object.

By default, `PyRegister` registers the name of the Python function with Mrm as the UIL-declarable callback function. An optional third string argument to PyRegister can be used to explicitly specify the UIL-declarable callback name registered with Mrm.

Currently, the internal implementation of PyRegister is ugly and only supports registering 250 routines.

There is one UIL-declarable function:

`PyEval(`*string*`)`

This UIL-declarable-function is a Mrm-registered callback function that accepts a string argument that it evaluates in the context of the Python `__main__` module. `PyEval` is for use in UIL, not in Python code. Any non-indented Python code is legal as the string argument; semi-colons can be used to construct a multi-statement argument. Beware of spaces, which are significant. The string argument originates in the uil file as the tag argument to the `PyEval` function.

An example of using PyEval in UIL code is:

```
procedure PyEval(string);
...
XmNactivateCallback = procedure PyEval("MyPrint(g_var)");
```

Before evaluating the string argument, `PyEval` adds two special symbols to the namespace of the Python `__main__` module. `__widget__` is the widget argument to the actual Mrm-registered `PyEval` callback function wrapped as a Python widget object. `__calldata__` is the calldata argument to this function wrapped as a Python callback object.

2. Callback models

There are three callback models for associating Python callback logic with UIL-defined widgets. The Examples section below demonstrates all of the models. The models are:

`Xt.AddCallback` With this approach, Python initialization logic attaches widget callbacks to Python functions (and the UIL callback feature is not used). This is likely to be the most popular callback model; if you cannot know which model to use, then use this approach. Example code:

```
...
main = hier.FetchWidget("main", top);
main.NameToWidget("*button").AddCallback(
                   'activateCallback', pythonFunction, "context");
...
```

`PyRegister` `PyRegister` registers a Python function using `MrmRegister` so that the function will be called as specified in UIL callback statements. (This model imitates the manner in which UIL specifies callbacks to C functions and the C program registers the callbacks during initialization.) One use for this approach would be to prototype an application in Python and then convert to C code, keeping the same UIL.

For example:

```
mrm_hier = Mrm.OpenHierarchy(["reg.uid"])
mrm_hier.PyRegister(MyPrint, "widget");
```

`MyPrint`, now a Mrm-registered Python routine, can be called directly from UIL.

`PyEval` `PyEval` is a UIL-callable function (automatically-registered) that dynamically interprets its string argument as Python code. This approach allows the logic associated with each callback to be entirely specified in the UIL module. This approach is somewhat experimental. The `PyEval` function is described above. Example UIL code:

```
procedure PyEval(string);
...
XmNactivateCallback = procedure PyEval("MyPrint(g_var)");
```

The string "MyPrint(g_var)" is evaluated by the Python interpreter whenever the activate callback is invoked. Evaluating the string is efficient because `PyEval` caches the compiled Python code object the first time the string is evaluated and compiled. Subsequently, the code object can be evaluated without having to be compiled.

3. Examples

Four complete examples are presented that demonstrate several different techniques for using UIL with Python.

- The "Simple Button using `AddCallback`" example shows how to use UIL to build a widget hierarchy but use the Python `AddCallback` method to add callbacks to the UIL-created widgets.
- The "Simple Button using `PyEval`" example shows how to use the Mrm-registered function `PyEval()` in place of the Python `AddCallback` method to add callbacks to widgets directly from UIL. This illustrates a powerful feature of the Mrm binding—the ability to connect to predefined Python functions without writing any code.
- The "Simple Button using `PyRegister`" example shows the use of `PyRegister` to declare a Python function to be a Mrm-registered callback function. This capability is useful for prototyping in UIL where the prototype will eventually be ported—the callbacks are migrated from Python code to C or C++, but the UIL remains the same.
- The "Command Line Program Front End" example shows the use of the `MrmNcreateCallback` and the `__widget__` symbol to enter the Python widget objects for widgets created in UIL into a Python symbol dictionary.

3.1 Simple Button using AddCallback

This example demonstrates about the simplest possible example of using the Mrm module. There are three files required—a .uil file named button.uil, a .uid file named button.uid, and a Python script named testbutton.py.
button.uil contains the following text:

```
module button_test
version = 'V1.0'
names = case_sensitive
```

```
object main : XmBulletinBoard {     ! bulletin board parent
    arguments {
        XmNwidth = 200;
        XmNheight = 200;
    };
    controls {
        XmPushButton button;        ! main is parent of button
    };
};

object button : XmPushButton {      ! button is labelled Push Me
    arguments {
        XmNx = 0;
        XmNy = 0;
        XmNwidth = 100;
        XmNheight = 40;
        XmNlabelString = 'Push Me';
    };
};

end module;
```

button.uil is compiled to button.uid by the statement:

```
uil button.uil -o button.uid
```

testbutton.py (which must have execute permission) contains the following left-justified text:

```
#!/usr/local/bin/python

import os, sys, Mrm, Xt

def OnActivate(widget, client, call):
    print 'OnActivate:',widget,client

def main():
    top_level = Xt.Initialize()

    # fetch the hierarchy
    mrm_hier = Mrm.OpenHierarchy('button.uid')
    main_w = mrm_hier.FetchWidget('main', top_level)
    button_w = main_w.NameToWidget('button')
    button_w.AddCallback('activateCallback', OnActivate, 'Hi')

    # manage widgets, realize shell
    main_w.ManageChild()
```

```
        top_level.RealizeWidget()

        Xt.MainLoop()

    main()
```

The script is executed by the statement:

```
testbutton.py
```

When executed, the testbutton.py script first declares the function `OnActivate()`. `OnActivate` is a typical callback function, called with widget, client data and call data arguments. It prints its widget and client data argument. The call data argument is binary, hence ugly when printed.
Then the script declares the function `main()`. `main()` opens the hierarchy defined by button.uid, fetches the widget named "main", which is a bulletin board, and its child, a pushbutton. Next, `main()` uses `NameToWidget` to obtain a Python widget object for the button widget, and adds `OnActivate()` as the `XmNactivateCallback` function, passing "Hi" as the client data. `main()` then manages these widgets, realizes the shell, and enters the Xt Event Loop.
Finally, the script invokes `main()`, which executes the function `main()`, and generates the interface as described above.
When the user pushes the pushbutton (labelled "Push Me" by the `XmNlabelString` resource specified in the .uil file), the `OnActivate()` function is called with the string "Hi" as its client data. `OnActivate` prints "Hi" on stdout.

3.2 Simple Button using PyEval

This example demonstrates the use of `PyEval()` in UIL to evaluate a string that invokes a callback. If the callback were "pre-defined", then a UIL-writer would not have to write any Python code to use this technique, other than the string passed to `PyEval()` to evaluate.
There are three files required—a .uil file named button2.uil, a .uid file named button2.uid, and a Python script named testbutton2.py.
button2.uil contains the following text:

```
module button_test_2
version = 'V1.0'
names = case_sensitive

procedure
PyEval(string);                     ! declare PyEval

object main : XmBulletinBoard {     ! bulletin board parent
    arguments {
        XmNwidth = 200;
        XmNheight = 200;
    };
```

```
    controls {
        XmPushButton button;        ! main is parent of button
    };
};

object button : XmPushButton {      ! button is labelled Push Me
    arguments {
        XmNx = 0;
        XmNy = 0;
        XmNwidth = 100;
        XmNheight = 40;
        XmNlabelString = 'Push Me';
    };
    callbacks {
        XmNactivateCallback = procedure PyEval("MyPrint(g_var)");
    };
};

end module;
```

button2.uid is created by the uil compiler as noted in the first example.
testbutton2.py (which must have execute permission) contains the following left-justified text:

```
#!/usr/local/bin/python

import os, sys, Mrm, Xt

def MyPrint(msg):                   # MyPrint is defined in __main__
    print msg

def main():
    global g_var                    # g_var has global scope in __main__
    g_var = 'a global value'
    print 'g_var =',g_var

    top_level = Xt.Initialize()

    # fetch the hierarchy
    mrm_hier = Mrm.OpenHierarchy('button2.uid')
    main_w = mrm_hier.FetchWidget('main', top_level)

    # manage widgets, realize shell
    main_w.ManageChild()
    top_level.RealizeWidget()

    Xt.MainLoop()
```

```
main()
```

The script is executed by the statement:

```
testbutton2.py
```

When executed, the testbutton2.py script first declares the function `MyPrint()`. `MyPrint()` is in the scope of the Python "unnamed" module `__main__` because no other module was named for `MyPrint()` to be in the scope of. This is an important point, because `PyEval` evaluates its string in this namespace, and so can find `MyPrint()`.
Then the script declares the function `main()`. `main()` declares the global `g_var`. `g_var` is also in the `__main__` namespace, for the same reason as `MyPrint()`. `g_var` is defined in this namespace so that it can be used in the tag argument of the `PyEval` call. The script opens the hierarchy defined by button.uid, fetches the widget named "main", which is a bulletin board, and its child, a pushbutton, manages these widgets and realizes the shell, and enters the Xt Event Loop.
Finally, the script invokes the `main()` function, which executes the code and builds the interface.
When the user pushes the pushbutton (labelled "Push Me" in the .uil file), the `PyEval()` function is called with the string "MyPrint(g_var)". This string is evaluated by Python in the context of module `__main__`, and calls `MyPrint()` passing as the argument the variable `g_var`.
`MyPrint()` simply prints it's argument's value, which is that of `g_var`—"a global value".
Note that if the testbutton2.py script was imported, then `MyPrint()` would not be in `__main__` namespace. testbutton2.py could be imported by the statement:

```
import testbutton2
```

In this case, `MyPrint` and `g_var` would be globals in the module testbutton2's namespace; and so the .uil file would have to be coded differently to invoke `MyPrint()`, as shown below:

```
callbacks {
    XmNactivateCallback = procedure PyEval(
        "import testbutton2; testbutton2.MyPrint(testbutton2.g_var)");
};
```

3.3 Simple Button Using PyRegister
This example demonstrates the use of `PyRegister()` in UIL to register a Python function as an Mrm callback.
There are three files required—a .uil file named reg.uil, a .uid file named reg.uid, and a Python script named testreg.py.
reg.uil contains the following text:

```
module BX
version = 'V1.0'
names = case_sensitive
```

```
procedure MyPrint(widget);

object main : XmBulletinBoard {
    arguments {
        XmNwidth = 200;
        XmNheight = 200;
    };
    controls {
        XmPushButton button;
    };
};

object button : XmPushButton {
    arguments {
        XmNx = 0;
        XmNy = 0;
        XmNwidth = 100;
        XmNheight = 40;
        XmNlabelString = 'PushMe';
    };
    callbacks {
        XmNactivateCallback = procedure MyPrint(main);
    };
};

end module;
```

reg.uid is created by the uil compiler as noted in the first example.
testreg.py (which must have execute permission) contains the following left-justified text:

```
#!/usr/local/bin/python

import os, sys, Mrm, Xt

def MyPrint(widget, client, call):
    print 'MyPrint: widget =',widget,'client =',client

def main():
    top_level = Xt.Initialize()

    mrm_hier = Mrm.OpenHierarchy(['reg.uid'])
    mrm_hier.PyRegister(MyPrint, 'widget')
    main_w = mrm_hier.FetchWidget('main', top_level)

    main_w.ManageChild()
```

```
        top_level.RealizeWidget()

        Xt.MainLoop()

    main()
```

The script is executed by the statement:

```
testreg.py
```

When executed, the testreg.py script first declares the function `MyPrint()`. `MyPrint` is a typical Python callback function, and has widget, clientdata, and calldata arguments. `MyPrint` simply prints its widget and clientdata arguments.
Then the script declares the function `main()`. `main` opens the hierarchy defined by button.uid. `main()` then uses `PyRegister()` to register `MyPrint` as an Mrm callback function, with a client argument of type widget. Note that this must be done before the `FetchWidget` call, so that `MyPrint()` is registered before being used in the UIL. `main()` then fetches the widget named "main", which is a bulletin board, and its child, a pushbutton, manages these widgets and realizes the shell, and enters the Xt Event Loop.
Finally, the script invokes the `main()` function, which executes the code and builds the interface.
When the user pushes the pushbutton (labelled "Push Me" in the .uil file), the `MyPrint()` function is called. The widget argument is the pushbutton widget wrapped as a Python widget object; the client argument is the bulletin board widget main wrapped as a Python widget object; and the call argument is the call data structure appropriate to an `XmNactivateCallback` wrapped as a Python string object. `MyPrint()` prints the widget and call arguments.
Because `MyPrint` was declared to have a client data argument of type widget, when `MyPrint` is called the client argument is wrapped as a Python widget argument. The argument could also have been declared to be of type integer or string, in which case it would have be wrapped as the appropriate Python object. The declaration of the argument type of `MyPrint()` in the `PyRegister` function should match that declared in UIL.

3.4 Command Line Program Front End

This example is a simple front end to the unix "cp" command that demonstrates the use of the `MrmNcreateCallback` to enter the widgets created by UIL into module `__main__`'s dictionary. This is somewhat cleaner than having to find the UIL-created widgets using `NameToWidget`. The files involved are panel_cp.uil, panel_cp.uid and panel_cp.py.
panel_cp.uil contains the following:

```
module cp_front_end
version = 'V1.0'
names = case_sensitive

procedure PyEval(string);

object main : XmBulletinBoard {
```

```
    arguments {
        XmNwidth = 300;
        XmNheight = 400;
    };
    controls {
        XmText src_file;
        XmText dst_file;
        XmPushButton button;
    };
};

object  src_file : XmText {
    arguments {
        XmNy = 0;
        XmNx = 10;
        XmNwidth = 280;
        XmNheight = 40;
        XmNeditMode = XmSINGLE_LINE_EDIT;
    };
    callbacks {
        MrmNcreateCallback = procedure PyEval("OnCreate('src_file_w')");
        XmNvalueChangedCallback = procedure PyEval("OnValueChanged()");
    };
};

object  dst_file : XmText {
    arguments {
        XmNy = 60;
        XmNx = 10;
        XmNwidth = 280;
        XmNheight = 40;
        XmNeditMode = XmSINGLE_LINE_EDIT;
    };
    callbacks {
        MrmNcreateCallback = procedure PyEval("OnCreate('dst_file_w')");
        XmNvalueChangedCallback = procedure PyEval("OnValueChanged()");
    };
};

object  button : XmPushButton {
    arguments {
        XmNy = 110;
        XmNx = 10;
        XmNwidth = 100;
        XmNheight = 40;
```

```
        XmNlabelString = 'Do Copy';
        XmNsensitive = false;
    };
    callbacks {
        MrmNcreateCallback = procedure PyEval("OnCreate('button_w')");
        XmNactivateCallback = procedure PyEval("DoCopy()");
    };
};

end module;
```

panel_cp.uil is compiled to panel_cp.uid by the statement:

```
uil panel_cp.uil -o panel_cp.uid
```

panel_cp.py contains the following text:

```
#!/usr/local/bin/python

import posix, os, sys, Mrm, Xm, Xt

# -----------------------------------------
def OnCreate(name):
    # insert name=__widget__ into __main__'s dictionary
    sys.modules['__main__'].__dict__[name] = __widget__

# -----------------------------------------
def OnValueChanged():
    if len(src_file_w.TextGetString()) > 0 and \
       len(dst_file_w.TextGetString()) > 0:
        button_w.sensitive = 1
    else:
        button_w.sensitive = 0

# -----------------------------------------
def DoCopy():
    cmd = 'cp '
    cmd = cmd + src_file_w.TextGetString()
    cmd = cmd + ' '
    cmd = cmd + dst_file_w.TextGetString()

    print cmd
    posix.system(cmd)
    sys.exit()

# -----------------------------------------
```

```
def main():
    top_level = Xt.Initialize()

    mrm_hier = Mrm.OpenHierarchy("panel_cp.uid")
    main_w = mrm_hier.FetchWidget("main", top_level)

    main_w.ManageChild()
    top_level.RealizeWidget()

    Xt.MainLoop()

main()
```

This example demonstrates using the `OnCreate` `MrmNcreateCallback` to access the `__widget__` symbol and insert an entry in the `__main__` module's dictionary for the `XmText` and `XmPushButton` widgets declared in the UIL. This makes these widgets accessible as Python-wrapped widget objects to the `OnValueChanged` and `OnCopy` callback routines, without having to pass the widgets to these callbacks in the UIL tag argument.
Another way to accomplish this is to insert the names into the `__main__` namespace explicitly in `main()`, for example:

```
global src_file_w, dst_file_w

...
main_w = mrm_hier.FetchWidget("main", top_level)
src_file_w = main_w.NameToWidget('src_file')
dst_file_w = main_w.NameToWidget('dst_file')
```

Here, 'src_file' and 'dst_file' are the widget's names as specified in the UIL, and `NameToWidget` is used to find the widgets, obtain Python wrapped widget objects, and initialize src_file_w and dst_file_w global variables.

Chapter 7
Built-in module Xaw

1. Xaw functions and variables

The module defines the following variables:

`Clock`
  Widget class object.

`Command`
  Widget class object.

`Dialog`
  Widget class object.

`Form`
  Widget class object.

`Grip`
  Widget class object.

`Label`
  Widget class object.

`List`
  Widget class object.

`Logo`
  Widget class object.

`Mailbox`
  Widget class object.

`MenuButton`
  Widget class object.

`Paned`
  Widget class object.

`Panner`
  Widget class object.

`Porthole`
: Widget class object.

`Repeater`
: Widget class object.

`Scrollbar`
: Widget class object.

`Simple`
: Widget class object.

`SimpleMenu`
: Widget class object.

`StripChart`
: Widget class object.

`Text`
: Widget class object.

`Toggle`
: Widget class object.

`Tree`
: Widget class object.

`Viewport`
: Widget class object.

Chapter 8
Built-in module Glx

1. Glx functions and variables

The module defines the following functions and variables:

`Draw`
: Widget class object

`makeconfig(`*list*`)`
: Returns `int`.

`winset(`*glx_draw_widget*`)`
: Returns `None`. This function corresponds with `GLXwinset`.

Chapter 9
Built-in module HTML

1. HTML functions and variables

The module defines the following variable:

`html`
: Widget class object.

2. HTML Widget object

Widget objects whose widget class is subclassed from html have the following additional methods:

`AnchorToId(`*name*`)`
: Returns `id`. This function corresponds with `HTMLAnchorToId`. Convenience function to return the element id of the anchor based on the anchor NAME passed. Function returns id on success. If there is no such element, 0 is returned.

`AnchorToPosition(`*name*`)`
: Returns `(success, x, y)`. This function corresponds with `HTMLAnchorToPosition`. Convenience function to return the position of the anchor based on the anchor NAME passed. If there is no such element, x=0, y=0 and success=-1 is returned.

`ClearSelection()`
: Returns `None`. This function corresponds with `HTMLClearSelection`.

`FreeImageInfo()`
: Returns `None`. This function corresponds with `HTMLFreeImageInfo`.

`GetHRefs()`
: Returns `href_list`. This function corresponds with `HTMLGetHRefs`. Convenience function to return the HREFs of all active anchors in the document. Function returns a list of strings.

`GetImageSrcs()`
: Returns `srcs_list`. This function corresponds with `HTMLGetImageSrcs`. Convenience function to return the SRCs of all images in the document. Function returns a list of

strings.

`GetLinks()`

Returns `link_list`. This function corresponds with `HTMLGetLinks`. Convenience function to return the link information for all the ¡LINK¿ tags in the document. Function returns a list of tuples consisting of two strings, the `href` and `role` elements of a LinkInfo struct.

`GetText(`*pretty* [, *url*] `)`

Returns `text`. This function corresponds with `HTMLGetText`. Convenience function to return the text of the HTML document as a plain ascii text string. Extra option flags "pretty" text to be returned. When pretty is two or larger, Postscript is returned. The font used is encoded in the pretty parameter: pretty = 2: Times; pretty = 3: Helvetica; pretty = 4: New century schoolbook; pretty = 5: Lucida Bright

`GetTextAndSelection()`

Returns `(text, start, end, insert)`. This function corresponds with `HTMLGetTextAndSelection`. Convenience function to return the text of the HTML document as a single white space separated string, with pointers to the various start and end points of selections.

`GotoId(`*element_id*`)`

Returns `None`. This function corresponds with `HTMLGotoId`. Convenience function to position the element based on the element id passed at the top of the viewing area. A passed in id of 0 means goto the top.

`IdToPosition(`*element_id*`)`

Returns `(success, x, y)`. This function corresponds with `HTMLIdToPosition`. Convenience function to return the position of the element based on the element id passed in. If there is no such element, x=0, y=0 and success=-1 is returned.

`PositionToId(`*x*`, `*y*`)`

Returns `element_id`. This function corresponds with `HTMLPositionToId`. Convenience function to return the element id of the element nearest to the x,y coordinates passed in. If there is no element there, return the first element in the line we are on. If we are on no line, either return the beginning, or the end of the document.

`SearchText(`*pattern*`, `*start*`, `*backward*`, `*caseless*`)`

Returns `(page, (start_id, start_pos), (end_id, end_pos))`. This function corresponds with `HTMLSearchText`. Convenience function to search the text of the HTML document as a single white space separated string. Linefeeds are converted into spaces.

Takes a pattern, and the location to start the search from for incremental searching. Default start is the beginning of the document for forward searching, and the end of the document for backwards searching. The backward and caseless parameters I hope are self-explanatory.

Returns 1 on success (and the start and end positions of the match). Returns -1 otherwise.

`SetSelection(`*(start_id*`, `*start_pos), (end_id, end_pos)*`)`

Returns `None`. This function corresponds with `HTMLSetSelection`. Set the current selection based on the ElementRefs passed in. Both refs must be valid.

`SetText`(*text*, *header_text*, *footer_text* [, *element_id*, *target_anchor*])

Returns `None`. This function corresponds with `HTMLSetText`. Convenience function to set the raw text into the widget. Forces a reparse and a reformat. If any argument is passed in as None that text is unchanged, if an argument is an empty string, that text is set to NULL; Also pass an element ID to set the view area to that section of the new text. Finally pass an anchor NAME to set position of the new text to that anchor.

Index