Data Transfer with UTM

Motif permits experienced application programmers to extend or override the data transfer capabilities of many Motif widgets. However, data transfer is typically the responsibility of widgets rather than applications. For this reason, complete details of UTM data transfer are found in the *Motif Widget Writer's Guide*. The UTM information in that book is a prerequisite for understanding this chapter. Nevertheless, we do begin this chapter with a brief overview of UTM from the application programmer's perspective. After this overview, the chapter suggests a few relatively straightforward ways in which an application might use UTM.

What is UTM?

Motif widgets support none, some, or all of the following data transfer mechanisms:

- Primary transfer
- Secondary transfer
- Clipboard transfer
- Drag and drop

The Uniform Transfer Method (UTM) is not the fifth mechanism. Rather, UTM unifies and simplifies the way in which the four mechanisms are implemented. For example, consider an application that uses UTM to implement primary transfer. To add support for a UTM clipboard transfer, the application programmer need only write a little extra code. That is because UTM makes it easy for the different data transfer mechanisms to share code.

As of Motif Release 2.0, all widgets and applications that implement data transfer should use UTM for that implementation. As of Motif Release 2.0, the Motif toolkit implements data transfer exclusively through UTM.

Why Use UTM?

Most applications will be satisfied with the supported target lists. However, some applications may want to support additional targets. In fact, this is the primary reason for using UTM.

Consider the **XmDrawingArea** widget. This widget does not know how to convert any targets. Therefore, the standard **XmDrawingArea** cannot serve as a useful source for data transfer. However, **XmDrawingArea** does provide a UTM infrastructure. This infrastructure allows an application programmer to use UTM to supplement the data transfer capabilities of **XmDrawingArea**. For example, an application could supplement **XmDrawingArea** such that a user could copy text from an **XmText** widget to an **XmDrawingArea** widget. Similarly, an application could supplement **XmDrawingArea** such that a user could copy a pixmap from one **XmDrawingArea** to another.

Ideas For Applications

Your application can use two new callbacks, **XmNconvertCallback** and **XmNdestinationCallback**, to extend or override the data transfer capabilities of any Motif widget. In order to override the data transfer capability of a widget, an application has to provide the same data transfer code that a widget would. So, if those are your intentions, we refer you back to the *Motif Widget Writer's Guide* for complete details.

If, however, your intention is to extend the data transfer capabilities of a widget, then the section you are now reading should be helpful. The following suggest several ways in which an application could extend the data transfer capabilities of a widget:

- Your application can extend the list of targets supported by a widget. For example, perhaps your application can support a target called *_FORMATTED_TEXT*, which transfers text in some kind of justified formatted.
- Your application can extend the widget to support a new form of data transfer. For example, perhaps your application could extend a particular widget so that it supports Drag and Drop.
- Your application can provide a customized data transfer that improves runtime performance. For example, Motif toolkit widgets typically transfer only one target at a time. If your application always needs to transfer the same group of three targets, your application could request a special multiple transfer.

Before undertaking a UTM project in your application, you should consider the following:

- Data transfer code is relatively complicated to write.
- By extending a widget, you may run into some consistency problems. For example, suppose your application extends the data transfer capabilities of an **XmLabel** widget. Since other applications probably will not extend **XmLabel** in the same way yours has, you run the risk of confusing users. Your application's documentation will have to explain the ways in which **XmLabel** widgets in your application are different from standard **XmLabel** widgets.

Implementation Overview

This section provides an overview of UTM from an application programmer's perspective. For complete details, see the *Motif Widget Writer's Guide*. This section explores the following three kinds of routines that an application might write in order to implement a UTM data transfer:

- An **XmNconvertCallback** procedure associated with the source widget
- An **XmNdestinationCallback** procedure associated with the destination widget
- A transfer procedure associated with the **XmNdestinationCallback** procedure

In brief, an **XmNdestinationCallback** procedure requests specific targets. An **XmNconvertCallback** procedure converts selections to the requested targets. A transfer procedure pastes the converted selections into the destination widget.

XmNconvertCallback Procedure

In order to be the source of a UTM data transfer, a widget must provide a callback resource named **XmNconvertCallback**. The following Motif toolkit widgets hold this resource:

- **XmPrimitive** and all its subclasses
- **XmContainer**
- **XmDrawingArea**
- **XmScale**

Your application can attach an **XmNconvertCallback** procedure to any widget holding the **XmNconvertCallback** resource. UTM calls any **XmNconvertCallback** procedures in your application whenever the associated widget is asked to convert a selection. After calling these callbacks, UTM typically calls the widget's own internal conversion routine (the *convertProc* trait method of the **XmQTtransfer** trait)

When UTM calls your application's **XmNconvertCallback** procedures, UTM passes a pointer to a **XmConvertCallbackStruct** as the *client_data* argument. This callback structure is defined as follows:

```
typedef struct
{
        int      reason ;
        XEvent  *event ;
        Atom selection ;
        Atom target ;
        XtPointer source_data  ;
        XtPointer location_data  ;
        int flags ;
        XtPointer parm ;
        int parm_format  ;
        unsigned long parm_length  ;
        Atom parm_type  ;
        int status  ;
        XtPointer value ;
        Atom type ;
        int format ;
        unsigned long length ;
} XmConvertCallbackStruct;
```

reason

Indicates why the callback was invoked.

event

Points to the **XEvent** that triggered the callback. It can be NULL.

selection

Indicates the selection for which conversion is being requested. Possible values are **CLIPBOARD**, **PRIMARY**, **SECONDARY**, and **_MOTIF_DROP**.

target

Indicates the target to convert.

source_data

Contains information about the selection source. When the selection is **_MOTIF_DROP**, *source_data* is the DragContext. Otherwise, *source_data* is NULL.

location_data

Contains information about the location of data to be converted. If the value is NULL, the data to be transferred consists of the widget's current selection. Otherwise, the type and interpretation of the value are specific to the widget class.

flags

Indicates the status of the conversion. Following are the possible values:

XmCONVERTING_NONE

This flag is currently unused.

XmCONVERTING_PARTIAL

The target widget was able to be converted, but some data was lost.

XmCONVERTING_SAME

The conversion target is the source of the data to be transferred.

XmCONVERTING_TRANSACT

This flag is currently unused.

parm

Contains parameter data for this target. If no parameter data exists, the value is NULL.

When *selection* is **CLIPBOARD** and *target* is **_MOTIF_CLIPBOARD_TARGETS** or **_MOTIF_DEFERRED_CLIPBOARD_TARGETS**, the value is the requested operation (**XmCOPY**, **XmMOVE**, or **XmLINK**).

parm_format

Specifies whether the data in *parm* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 0 (when *parm* is NULL), 8 (when the data in *parm* should be viewed as a list of *char*s), 16 (when the data in *parm* should be viewed as a list of *short*s), or 32 (when the data in *parm* should be viewed as a list of *long*s). Note that *parm_format* symbolizes a data type, not the number of bits in each list element. For example, on some machines, a *parm_format* of 32 means that the data in *parm* should be viewed as a list of 64−bit quantities, not 32−bit quantities.

parm_length

Specifies the number of elements of data in *parm*, where each element has the size specified by *parm_format*. When *parm* is NULL, the value is 0.

parm_type

Specifies the parameter type of *parm*.

status

An IN/OUT member that specifies the status of the conversion. The initial value is **XmCONVERT_DEFAULT**. The callback procedure can set this member to one of the following values:

XmCONVERT_DEFAULT

This value means that the widget's *convertProc* trait method, if any, is called after the callback procedures return. If the widget's *convertProc* trait method produces any data, it overwrites the data provided by the callback procedures in the *value* member.

XmCONVERT_MERGE

This value means that the widget's *convertProc* trait method, if any, is called after the callback procedures return. If the widget's *convertProc* trait method produces any data, it appends its data to the data provided by the callback procedures in the *value* member. This value is intended for use with targets that result in lists of data, such as **TARGETS**.

XmCONVERT_DONE

This value means that the callback procedure has successfully finished the conversion. The widget's *convertProc* trait method, if any, is not called after the callback procedures return.

XmCONVERT_REFUSE

This value means that the callback procedure has terminated the conversion process without completing the requested conversion. The widget's *convertProc* trait method, if any, is not called after the callback procedures return.

value

An IN/OUT parameter that contains any data that the callback procedure produces as a result of the conversion. The initial value is NULL. If the callback procedure sets this member, it must ensure that the *type*, *format*, and *length* members correspond to the data in *value*. The callback procedure is responsible for allocating memory when it sets this member. The toolkit frees this memory when it is no longer needed.

type

An IN/OUT parameter that indicates the type of the data in the *value* member. The initial value is **INTEGER**.

format

An IN/OUT parameter that specifies whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. The initial value is 8. The callback procedure can set this member to 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).

length

An IN/OUT member that specifies the number of elements of data in *value*, where each element has the size symbolized by *format*. The initial value is 0.

If your application does not define any **XmNconvertCallback** procedures, UTM automatically calls the widget's *convertProc* trait method. If your application does define at least one **XmNconvertCallback** procedures, UTM conditionally calls the widget's *convertProc* trait method. Here are the conditions:

- If your **XmNconvertCallback** procedures are taking full responsibility for converting the selection to a particular target, then UTM will not call the widget's *convertProc* trait method. An **XmNconvertCallback** procedure notes that it is taking full responsibility by setting the *status* member of the **XmConvertCallbackStruct** to **XmCONVERT_DONE** or **XmCONVERT_REFUSE**.
- If your **XmNconvertCallback** procedures are not taking full responsibility for converting the selection to a particular target, then UTM will call the widget's *convertProc* trait method. An **XmNconvertCallback** procedure notes that it is not taking full responsibility by setting the *status* member of the **XmConvertCallbackStruct** to **XmCONVERT_DEFAULT** or **XmCONVERT_MERGE**.

If UTM does call your widget's *convertProc* trait method, UTM passes it the same **XmConvertCallbackStruct** used by the application's **XmNconvertCallback** procedures. Thus, the **XmConvertCallbackStruct** provides a convenient conduit for the **XmNconvertCallback** procedures to communicate with the widget's *convertProc* trait method.

XmNdestinationCallback Procedure

In order to be the destination for a UTM data transfer, a widget must provide a callback resource named **XmNdestinationCallback**. Appendix B lists the widgets in the standard Motif toolkit that hold this callback resource.

Your application can attach an **XmNdestinationCallback** procedure to any widget holding the **XmNdestinationCallback** resource. UTM calls any **XmNdestinationCallback** procedures in your application whenever the associated widget is the destination of a data transfer. After calling

these callbacks, UTM typically calls the destination widget's own internal destination routine (the *destinationProc* trait method of the **XmQTtransfer** trait).

When UTM calls your application's **XmNdestinationCallback** procedures, UTM passes a pointer to a **XmDestinationCallbackStruct** as the *client_data* argument. This callback structure is defined as follows:

```
typedef struct
{
        int      reason ;
        XEvent   *event ;
        Atom selection ;
        XtEnum operation ;
        int flags;
        XtPointer transfer_id ;
        XtPointer destination_data  ;
        XtPointer location_data ;
        Time time;
} XmDestinationCallbackStruct;
```

reason

Indicates why the callback was invoked.

event

Points to the **XEvent** that triggered the callback. It can be NULL.

selection

Indicates the selection for which data transfer is being requested. Possible values are **PRIMARY**, **SECONDARY**, **CLIPBOARD**, and **_MOTIF_DROP**.

operation

Indicates the type of transfer operation requested.

When the *selection* is **PRIMARY** or **SECONDARY**, possible values are **XmMOVE**, **XmCOPY**, and **XmLINK**.

When the selection is **CLIPBOARD**, possible values are **XmCOPY** and **XmLINK**.

When the selection is **_MOTIF_DROP**, possible values are **XmMOVE**, **XmCOPY**, **XmLINK**, and **XmOTHER**. A value of **XmOTHER** means that the callback procedure must get further information from the **XmDropProcCallbackStruct** in the *destination_data* member.

flags

Indicates whether or not the destination widget is also the source of the data to be transferred. Following are the possible values:

XmCONVERTING_NONE

The destination widget is not the source of the data to be transferred.

XmCONVERTING_SAME

The destination widget is the source of the data to be transferred.

transfer_id

Serves as a unique ID to identify the transfer transaction.

destination_data

Contains information about the destination. When the selection is **_MOTIF_DROP**, the callback procedures are called by the drop site's **XmNdropProc**, and *destination_data* is a pointer to the **XmDropProcCallbackStruct** passed to the **XmNdropProc** procedure. When the selection is **SECONDARY**, *destination_data* is an Atom representing a target recommended by the selection owner for use in converting the selection. Otherwise, *destination_data* is NULL.

location_data

Contains information about the location where data is to be transferred. The interpretation of this value varies considerably from widget to widget. Once **XmTransferDone** procedures start to be called, *location_data* will no longer be stable.

time

Indicates the time when the transfer operation began.

The fields in an **XmConvertCallbackStruct** are far more dynamic than the fields in an **XmDestinationCallbackStruct**. In other words, an **XmNconvertCallback** procedure typically modifies many of the fields of its **XmConvertCallbackStruct**; however, an **XmNdestinationCallback** procedure typically leaves all the fields of its **XmDestinationCallbackStruct** untouched.

A typical **XmNdestinationCallback** procedure requests a list of targets supported by the source widget. To make this request, the procedure calls **XmTransferValue**. For example, here is a typical **XmNdestinationCallback** procedure:

```
void DestinationCallback(w, ignore, cs)
       Widget w;
       XtPointer ignore;
       XmDestinationCallbackStruct *cs;
{
 Atom TARGETS = XInternAtom(XtDisplay(w), "TARGETS", False);

 /* Request TARGETS that the source widget can convert. */
   XmTransferValue(cs -> transfer_id, TARGETS,
                   (XtCallbackProc) TransferProcedure,
                   NULL, XtLastTimestampProcessed() );
}
```

Transfer Procedure

The transfer procedure is a companion routine to the **XmNdestinationCallback** procedure. Typically,

1.The **XmNdestinationCallback** procedure calls **XmTransferValue** to ask for a list of targets supported by the source.
2.When the source completes the conversion, UTM returns control to the transfer procedure.
3.The transfer procedure examines the returned list of targets. If the list contains a desired target, the transfer procedure requests it by calling **XmTransferValue**.

Steps 2 and 3 may repeat several times.

Unlike the **XmNconvertCallback** and **XmNdestinationCallback** routines, the transfer procedure is not identified by a resource. Rather, the transfer procedure is identified by the third argument in a call to **XmTransferValue**.

When UTM calls your application's transfer procedure, UTM passes a pointer to an **XmSelectionCallbackStruct** as the *client_data* argument. This callback structure is defined as follows:

```
typedef struct
{
        int      reason ;
        XEvent  *event ;
        Atom selection ;
        Atom target ;
        Atom type ;
        XtPointer transfer_id ;
        int flags ;
        int remaining ;
        XtPointer value ;
        unsigned long length ;
        int format ;
} XmSelectionCallbackStruct;
```

reason

Indicates why the callback was invoked.

event

Points to the **XEvent** that triggered the callback. It can be NULL.

selection

Specifies the selection that has been converted.

target

Specifies the target to which **XmTransferValue** requested conversion. The value is the same as the value of the *target* argument to **XmTransferValue**.

type

Specifies the type of the selection value. This is not the target, but the type used to represent the target. The value **XT_CONVERT_FAIL** means that the selection owner did not respond to the conversion request within the Intrinsics selection timeout interval.

transfer_id

Specifies a unique indentifier for the data transfer operation. The value is the same as the value of the *transfer_id* argument to **XmTransferValue**.

flags

This member is currently unused. The value is always **XmSELECTION_DEFAULT**.

remaining

Indicates the number of transfers remaining for the operation specified by *transfer_id*.

value

Represents the data transferred by this request. The application is responsible for freeing the value by calling **XtFree**.

length

Indicates the number of elements of data in *value*, where each element has the size symbolized by *format*. If *value* is NULL, *length* is 0.

format

Indicates whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).

Case Study: Adding an Extra Target to XmText

This section illustrates how an application can use UTM to supplement the list of targets that an **XmText** widget can convert.

Here is the scenario. The application instantiates two **XmText** widgets. We will refer to one of these widgets as *TextSource* and the other as *TextDestination. TextDestination* has a rather unusual requirement; namely, it expects that any text transferred to it must not contain any lowercase letters. In other words, the source of a data transfer is responsible for converting any lowercase letters to uppercase before transferring the data.

The standard **XmText** widget does not provide any targets that can do this conversion. Therefore, the application must temporarily extend **XmText** to handle a new target named **MYTEXT**. The application must do the following:

- The application must associate an **XmNconvertCallback** procedure with *TextSource*. This procedure must be able to convert the **MYTEXT** target.
- The application must associate an **XmNdestinationCallback** procedure and a transfer procedure with *TextDestination*. These procedures are responsible for requesting that the selection be converted to **MYTEXT** and for pasting the transferred data into *TextDestination*.

The application will support three different forms of data transfer: primary, clipboard, and drag and drop.

The following subsections explains how the application accomplishes its goals.

The XmNconvertCallback Procedure

The application must associate an **XmNconvertCallback** procedure with widget *TextSource*. The following code does just that:

```
XtAddCallback(TextSource, XmNconvertCallback, ConvertCallback, NULL);
```

The application must provide an **XmNconvertCallback** procedure named *ConvertCallback*. This procedure must handle the following requests:

- The destination could request the list of targets that the source can convert.
- The destination could request that the selection be converted to **MYTEXT**.

Following are the relevant parts of the *ConvertCallback* routine:

```
void
ConvertCallback(Widget  w,
                    XtPointer ignore,
                    XtPointer call_data)
{
 XmConvertCallbackStruct  *ccs = (XmConvertCallbackStruct *)call_data;
 char    *selected_text;
 char    *copy_of_selected_text;
 Atom TARGETS = XInternAtom(XtDisplay(w), "TARGETS", False);
 Atom _MOTIF_CLIPBOARD_TARGETS = XInternAtom(XtDisplay(w),
                            "_MOTIF_CLIPBOARD_TARGETS", False);
 Atom MYTEXT = XInternAtom(XtDisplay(w), "MYTEXT", False);
 int   n=0;
 Atom *targs = (Atom *)XtMalloc(sizeof(Atom) * 2);

  if ((ccs->target == TARGETS) ||
      (ccs->target == _MOTIF_CLIPBOARD_TARGETS)) {

  /* Use targs to hold a list of targets that my application can
     convert. This list will be merged with the targets that the
     XmText widget can convert. */
    targs[n] = MYTEXT; n++;
    ccs->value = (XtPointer) targs;
    ccs->type = XA_ATOM;
    ccs->format = 32;
    ccs->length = n;
    ccs->status = XmCONVERT_MERGE;
  }
  else if (ccs->target == MYTEXT)  {
  /* Get the selection. */
```

```
    selected_text = XmTextGetSelection(w);
    copy_of_selected_text = selected_text;

  /* Convert any lowercase letters in the selection to uppercase. */
    while (*selected_text++)  {
       if (islower(*selected_text))
         *selected_text = toupper(*selected_text);
    }

  /* Place the converted text into the XmConvertCallbackStruct. */
    ccs->value = copy_of_selected_text;
    ccs->type = ccs->target;
    ccs->format = 8;
    ccs->length = strlen(copy_of_selected_text);
    ccs->status = XmCONVERT_DONE;
  }
}
```

Notice how *ConvertCallback* sets the value of the *status* member of the **XmConvertCallbackStruct**. For example, when the destination requests **TARGETS** or **_MOTIF_CLIPBOARD_TARGETS**, *ConvertCallback* sets *status* to **XmCONVERT_MERGE**. This *status* tells UTM to call the *convertProc* trait method of **XmText**. This trait method will add **MYTEXT** to the list of supported targets. As another example, consider that *ConvertCallback* is taking full responsibility to convert the selection to **MYTEXT**. For this case, *ConvertCallback* sets *status* to **XmCONVERT_DONE**. This status tells UTM not to bother calling the *convertProc* trait method of **XmText**. Finally, consider what happens when *ConvertCallback* is asked to convert a target other than **TARGETS**, **_MOTIF_CLIPBOARD_TARGETS**, or **MYTEXT**. In this case, the *status* member will retain its default value of **XmCONVERT_DEFAULT**. This *status* tells UTM to call the *convertProc* trait method of **XmText**, and that the trait method will be reponsible for converting the target.

The XmNdestinationCallback Procedure

The application must associate an **XmNdestinationCallback** procedure with widget *TextDestination*. The following code does just that:

```
XtAddCallback(TextDestination, XmNdestinationCallback,
              DestinationCallback, NULL);
```

The application must provide an **XmNdestinationCallback** procedure named *DestinationCallback*. We have implemented this procedure in a very basic fashion. This procedure merely requests a list of the targets that the source widget supports:

```
void
DestinationCallback(Widget  w,
                    XtPointer ignore,
                    XtPointer call_data)
{
 XmDestinationCallbackStruct *dcs =
          (XmDestinationCallbackStruct *)call_data;
 Atom TARGETS = XInternAtom(XtDisplay(w), "TARGETS", False);

 /* Ask the source to return a list of all the targets supported. */
   XmTransferValue(dcs->transfer_id, TARGETS,
                   (XtCallbackProc)TransferProc,
                   NULL, XtLastTimestampProcessed() );
}
```

The Transfer Procedure

When the source returns the list of **TARGETS**, UTM calls the application's transfer procedure. The transfer procedure, *TransferProc* in this case, is named by the third argument to **XmTransferValue**.

Here is the algorithm for the transfer procedure. The transfer procedure must examine the returned list of targets to see if the list contains **MYTEXT**. If it does, ask the source to convert the selection to **MYTEXT**. When the source completes the conversion, the transfer procedure must paste the converted text into the *TextDestination* widget.

Following is the code for *TransferProc*:

```
void
TransferProc(Widget  w,
             XtPointer ignore,
             XtPointer call_data)
{
 XmSelectionCallbackStruct *scs =
       (XmSelectionCallbackStruct *) call_data;
 Atom TARGETS = XInternAtom(XtDisplay(w), "TARGETS", False);
 Atom MYTEXT = XInternAtom(XtDisplay(w), "MYTEXT", False);
 Atom  *targets = (Atom *)scs->value;
 int    MYTEXT_is_supported = 0;
 unsigned long    n;


   if ((scs->target == TARGETS) && (scs->type == XA_ATOM))  {
     for (n=0; n<=scs->length; n++)  {
      /* Look through list of returned TARGETS to see if
         MYTEXT is there. */
        if (targets[n] == MYTEXT)
          MYTEXT_is_supported = 1;
     }

     if (MYTEXT_is_supported)
     /* Now ask the source widget to convert the selection
          to MYTEXT. */
       XmTransferValue(scs->transfer_id, MYTEXT,
                   (XtCallbackProc)TransferProc,
                   NULL, XtLastTimestampProcessed() );
   }

  if ((scs->target == MYTEXT)) {
    XmTextPosition current_insertion_position;
  /* Source widget has converted MYTEXT, paste it into the
     destination widget. */
    current_insertion_position = XmTextGetInsertionPosition(w);
    XmTextInsert(w, current_insertion_position, (char *)scs->value);
    XmTransferDone(scs->transfer_id, XmTRANSFER_DONE_SUCCEED);
  }
}
```

Notice how *TransferProc* calls **XmTransferDone**. This call marks the end of the transfer. Therefore, UTM will not call the *destinationProc* trait method inside the *TextDestination* widget.

If the source does not know how to convert **MYTEXT**, UTM will call the *destinationProc* trait method inside the *TextDestination* widget.