The DND Protocol
version 1.0

César Augusto Rorato Crusius

November 27, 1996

Contents

List of Tables

List of Examples

Chapter 1
Introduction

This document describes how to enable your `X` and `Xt` programs to use drag and drop using the DND protocol. If you program uses `Xt`, then a library is provided, and introducing drag and drop capabilities is a very simple task. If your program is `Xlib` based, then you will have to implement some things, but if your program does not start a drag, things are very simple too. If you want to see DND protocol in action, try `OffiX`, a collection of general utilities already using DND protocol (file manager and text editor included).

1.1 What is a drag and drop protocol ?

I'm supposing you understand what "drag and drop" means. It's an intuitive way to exchange data between programs. A drag and drop protocol is the mean used by the programs to communicate with each other in order to perform this data exchange. If the programs want to exchange information, they must be able to communicate with each other in a common language. This language is the "protocol." Therefore, if you create your own protocol, chances are other programs will not understand you. So you must "talk" to other programs in a common language. In this way, when you receive a drop, you will receive a "I dropped on you" message, and you will understand the message no matter who sended it to you. This is the purpose of a drag and drop protocol: to provide a common language for this kind of actions.

1.2 What is Dnd, and why should I use it ?

DND is a drag and drop protocol developed by César Crusius. The development began when the author tried to bring drag and drop capabilities to the file manager `xfm` in order to make a "trash can" program. The original idea was to use `rdd`, a drag and drop protocol created by Roger Reynolds. César tried then to get in contact with Roger to make sure the `rdd` protocol was still maintained, because if it wasn't César was inclined to take responsibility for the task. Due to a delay in the e-mail answer,

César decided to build a new protocol, and then DND was born. Before the announcement of the protocol some demonstration programs were needed. At this point César joined forces with André Hentz, and the `OffiX` project was born.
If you have a program that could benefit from having drag and drop capabilities, DND might be a good choice for the following reasons:

- It is public, unlike other "standard" protocols such as those used by Open Look and Motif. These two are incompatible, and aren't, as far as I know, published in detail. Since Open Look and Motif applications are `Xt` based, you can choose to use DND or the specific protocol. This could be a compile-time option, for example.
- You can use this protocol with *any* `X` program. Motif, Open Look and Athena applications can exchange data using the DND protocol. Non-`Xt` programs can incorporate DND protocol too.
- A `C` `Xt` interface is provided, so you can add drag and drop capabilities to an existing `Xt` program with a minimum effort.

Of course, there are also reasons to *not* use DND:

- You *must* exchange data between your application and another that uses another protocol.
- You don't need drag and drop, and in this case I wonder why are you still reading this.

I encourage you to use DND at least as a compile-time option for your program. There is already a good reason[1] to do that: the "files" program furnished with `OffiX`. Won't it be good to be able to exchange data with a file manager? Certainly will! And, as you will note, it's easy to bring this new capability to your program.

1.3 Getting and installing Dnd

At the current stage you can only get the `Xt` interface for DND. If you want to use DND in non-`Xt` applications, you must write some code. It is not a difficult task to do so.
The best way to install the DND `Xt` interface library is the following:

[1] The DND protocol is becoming widely accepted. Here is a list of some programs that use the protocol (officially or with patches): XEmacs, WINE (Windows emulator for Linux), xdvi, Image Magick, xv, ghostview...

- Get the DND library distribution. It can be found in the WEB page `http://www.inf.ufrgs.br/~ahentz/OffiX`.
- Decompress the file under some directory. You will have now a directory `DND`, and two directories inside this one. Go to the `DND/DNDlib` directory and follow the `README` file guidelines. This will install the library and the header files.
- Now go to the `DND/DNDdoc` directory. There you will find the complete protocol documentation, including a reference guide. To print the documentation follow the guidelines of the `README` file.

Now you should have the static DND `C` and `C++` libraries (`libDnd` and `libDnd++` respectively) and the necessary header files installed on your system. If you are not root and want to compile your program with DND, just copy the files `libDnd.a` and the DND header files from the `OffiX` directory and compile and link them with your program[2] (pass the options `-L. -lDnd` to the linker). If your program is in `C++` use the `libDnd++.a` instead.

1.4 Dnd and window managers

Table 1.1 summarize the compatibility between DND and window managers. A window manager is *fully compatible* with DND if drops on icons work under the window manager. If you run DND programs under another window manager, please send me a note at `c.crusius@ieee.org` telling about the results.

Fully compatible	**Compatible**
FVWM 1.24 (*)	open windows
FVWM 2	mwm
FVWM95	
vtwm 5.3 (*)	
twm (*)	

Table 1.1: Window managers and DND. Entries marked with (*) must be patched to be fully compatible, and you can get the patches with the `OffiX` distribution.

[2] You are also able to install the files in other places – see the `README` file.

Chapter 2

The Xt interface

2.1 Introduction

When a drag or a drop action occur within your program certain functions will be called. These functions are called *event handlers*, because they are only called when certain *event* occurs. The format of an Xt event handler function is

void `EventHandler`(Widget, XtPointer, XEvent* , Boolean*);

Thanks to DND, that is all you have to know about events and event handlers. I will assume only that you know what a widget is (since you are programming with the Xt interface this is very fair). Your program may have a lot of event handlers, since you may want to handle drop events in different ways for different areas of your application. The same thing applies to drag actions.

2.2 Initializing Dnd

For each module of your program that uses DND you must include the header file `OffiX/DragAndDrop.h`. The next thing to do is to initialize DND. This is done with the function

void `DndInitialize`(Widget toplevel);

where `toplevel` is the top level widget of your application. You must call the function after the creation of the widget (e.g. after `XtAppInitialize`). The next thing to do is to register the widgets you want to be able to receive drops and/or to send drag events (note that a widget can be able to receive drops but nothing can be dragged from it - e.g. a waste basket). You can do this with the functions

void `DndRegisterDropWidget`(Widget widget,
XtEventHandler handler,
XtPointer d);

void `DndRegisterDragWidget`(Widget widget,
XtEventHandler handler,
XtPointer d);

where `widget` is the widget in consideration, `handler` is the function to be called when the drop or drag actions take place, and `d` is something you should not care about, just replace with NULL.
If you want to handle drops on non-registered widgets, use the function

void `DndRegisterOtherDrop`(XtEventHandler handler);

for drops on icons,

void `DndRegisterIconDrop`(XtEventHandler handler);

and, if something can be dragged from your program, you may want to handle drops on the root window (e.g. the "files" program open a new file window each time a directory is dropped on the root window),

void `DndRegisterRootDrop`(XtEventHandler handler);

Now your DND protocol is initialized, and you must define the handlers and link your program with the option `-lDnd` and everything should work. The next sections are devoted to the handler functions definition. Before going on, just a note about the icon drops:
Icon drops will not work with the vast majority of window managers. This happens because the managers do not pass the client message events that are sent to the icons to the client windows. If the window manager does that, then icon drops will work.

2.3 The Dnd data types

The current data types supported by DND are given in table 2.1.

Constant	Meaning
`DndRawData`	Any kind of data.
`DndFile`	A file name.
`DndFiles`	List of file names.
`DndText`	Text block.
`DndDir`	Directory name.
`DndLink`	Name of a stale link.
`DndExe`	Name of a program.
`DndUnknown`	Undefined data.
`DndURL`	An URL location.
`DndMIME`	Data with MIME header.

Table 2.1: DND data types for version 1

When you initialize the data (with the `DndSetData` function) you must furnish the correct structure for DND. The structure for the various types of data is the following:

DndRawData Any kind of data. If the data you want to send does not fall into any category, use this one. *Do not use DndUnknown.* See the `DndUnknown` description to see why.

DndFile A null-terminated string containing the name (with the complete path) of a file.

DndFiles A sequence of null-terminated strings, each one containing a complete file name. The end of the list is determined by a zero-length file name.

DndText A null-terminated text block containing valid text characters.

DndDir A null-terminated string containing the name (with the complete path) of a directory.

DndLink A null-terminated string containing the name (with the complete path) of a link that points to nowhere (a stale link).

DndExe A null-terminated string containing the name (with the complete path) of an executable file (a program).

DndURL A complete address of a URL location (null-terminated string).

DndMIME Data preceded by a MIME header.

DndUnknown You must not set this type to any kind of data. Use `DndRawData` instead. You can only *receive* unknown data. This can happen when a program compiled with a newer version of DND send a data that your version of DND does not understand. Your program must handle this kind of data as if it were a `DndRawData`.

2.4 Retrieving data: the drop event handler

Ok, so you received a drop, and the event handler for this drop was called. The first think you need to know, as a programmer, is that your drop event handler can be called even if a drop did not occur (it's a rare thing, but can happen). The next thing you need to know is the data type of the drop event. Both needs can be satisfied by one of the functions

int `DndIsDropMessage`(XEvent *event);

int `DndDataType`(XEvent *event);

where `event` is the event that the drop handler received. The function `DndIsDropMessage` returns non-zero only if the event was really a drop message. The function `DndDataType` returns `DndNotDnd` if the event was not a DND drop event, and the drop data type otherwise.
Ok, it's a drop event, and you want to process the specific data type. To retrieve the drop data, use the function

void `DndGetData`(unsigned char **Data, unsigned long *Size);

where `Data` is the data recipient and `Size` a unsigned long that will contain the data length (including NULL characters at the end of strings). An example of how to use these two functions in a drop event handler is given in example 2.1
Now, just a note about memory management: the actual version of DND does not reserve any space for the data. Therefore, if your data processing takes too long and the user make another drag and drop action the data will be corrupted. So, if you want to keep the data for a long time, the best thing to do is to allocate space for it and copy the returned contents.

Example 2.1 Retrieving data with the `Xt` interface

```
void MyDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int DataType;
    unsigned long DataSize;
    char *Data;

    if(DndDataType(event)==DndNotDnd) return;
    DndGetData(& Data,& DataSize);

    /* ... data processing ... */
}
```

2.5 Sending data: the drag event handler

Hey - the user said - I want to send data from you to another program right now! So here you are, on the drag event handler. You, as the programmer, wants to set the data and let DND take care of the rest. For the first task you can use the function

void `DndSetData`(int Type, unsigned char *Data , unsigned long Size);

The `Size` field must be the total size of the data, including the NULL character of strings. The `Data` must be formatted according to the data `Type`. For a description of the data structures refer to section 2.3.
Now all you need to do is to call DND and let it handle the drag action by calling the function `DndHandleDragging`. Example 2.2 shows a typical drag handler.
Once a drag action takes place, `DndHandleDragging` will take control and will return only after the corresponding drop. Now, two things worth reading:

- You don't need to allocate space for the Data, DND will do that for you. In fact, when you call `DndSetData`, the old data stored on the DND global buffer will be destroyed.

- The drag event handler can be called many times before a drag actually take place. The `DndSetData` function will return automatically

Example 2.2 Example of an Xt drag event handler

```
void MyDragEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int DataType;
    unsigned long DataSize;
    char *Data;

    /* ... data initialization ... */

    DndSetData(DataType,Data,DataSize);
    DndHandleDragging(w,event);
}
```

if the data is already set. But the handler will perform the data initialization anyway, be careful. The function `DndHandleDragging` will return non-zero only if a drag really took place, otherwise it will return zero. This can be useful if you want to keep track of the current drag state.

2.6 Motif issues

The only purpose of this section is to give some advice on using DND with other widget kits that already have drag and drop capabilities. If you only receive drops there will be no problems, but if you start a drag you may face a dubious situation: which protocol will be used by the program – the original one or the DND?

We give here the solution suggested by Fred Pesch for the Motif toolkit: you will have to disable the standard drag capabilities of Motif in order to get DND drags to work. You can do this using the following resources

```
Client*dragInitiatorProtocolStyle : DRAG_NONE
Client*dragReceiverProtocolStyle: DRAG_NONE
```

Chapter 3

The DND protocol

3.1 Simple description

In this section I will give the minimal information necessary for the non-Xt programmers to handle DND drops. DND defines two atoms that you must define too. This can be done with the following code:

```
Atom DndProtocol,DndSelection;
DndProtocol=XInternAtom(display,"_DND_PROTOCOL",False);
DndSelection=XInternAtom(display,"_DND_SELECTION",False);
```

In older versions (version 0) the atom strings were "DndProtocol" and "DndSelection", but this was not ICCCM compatible. If you want to receive drops from older versions you may want to check for these atoms too (see section 3.3 for more details).
When a drop occurs, DND will send a client message event to the top level window of the application that receives the drop. The event will have the following data:

```
Event.xclient.type      = ClientMessage;
Event.xclient.message_type  = DndProtocol;
Event.xclient.format        = 32;
Event.xclient.data.l[0]     = DataType;
Event.xclient.data.l[1]     = (long)event->xbutton.state;
Event.xclient.data.l[2]     = (long)window;
Event.xclient.data.l[3]     = (long)(drop_x + 65536*drop_y);
Event.xclient.data.l[4]     = 1; /* the protocol version */
```

So your program main event loop will need to intercept this message and take the appropriate action. Normally, this action consists in getting the drop data and processing it. This can be done by getting the DndSelection property contents of the root window (with the function XGetWindowProperty).

3.2 The drag protocol

When a program begins a drag, some actions must be performed until the drop occurs. Normally the following tasks will be performed:

1. The data of the drop is initialized.
2. The program grabs the mouse and changes its cursor.
3. The program waits for a button release event.
4. The drop event is prepared.
5. The drop event is sent.

These actions may not appear in this very order, but some things are important: when the program changes the mouse cursor, it is a good behavior to change it according to the type of data that is being sent. So step 1 must be present at least to set the data type. The actual data initialization can be performed anytime before step 5. This setting can be performed by setting the contents of the `DndSelection` property of the root window with the function `XChangeProperty`.
Now to step 2: the program should not grab the mouse and change its cursor immediately. Suppose you clicked in a toggle, for example in a file icon in a file manager. This does not mean you want to drag something, you may want simply to select that file. So the program must drag the mouse only after some movement is performed. In the `Xt` interface the program will wait for a 10-pixel movement before actually grabbing the mouse. In the `Xt` library the grab is performed by the following call to `XGrabPointer`:

```
XGrabPointer(dpy,root,False,
    ButtonMotionMask|ButtonPressMask|ButtonReleaseMask,
    GrabModeSync,GrabModeAsync,root,
    DndCursor[DataType].CursorID,
    CurrentTime);
```

Next, the program enter in a loop waiting for a button release event. The following code could perform this action (a simplified version of the `Xt` library code):

```
int Dragging=1;
while(Dragging)
{
    XAllowEvents(dpy,SyncPointer,CurrentTime);
    XtAppNextEvent(app,&Event);
    switch(Event.type)
    {
    case ButtonRelease:
        Dragging=0;
        break;
    default:
        XtDispatchEvent(&Event);
        break;
    }
}
```

The only "strange" thing is the call to the function `XAllowEvents`. This is necessary in order to queue enter and leave events for posterior processing by other clients. In the `Xt` implementation enter and leave events are used to update the widget into which the drop occurred.
After this loop the drop event must be prepared. The drop event a ClientMessage event, ant its structure is given in table 3.1

Event field	**Contents**
type	`ClientMessage`
message_type	`DndProtocol`
format	`32`
data.l[0]	`DataType`
data.l[1]	`(long)event->xbutton.state`
data.l[2]	`(long)window`
data.l[3]	`(long)(drop_x + 65536*drop_y)`
data.l[4]	protocol version

Table 3.1: Structure of the DND drop event

Next we send this event with `XSendEvent` to the appropriate window. But which is the appropriate window? It is the window under the mouse, of course, but how to find this window? In the `Xt` implementation this is done with the `Xmu` library function `XmuClientWindow` as follows

```
Target=XmuClientWindow(dpy,Event.xbutton.subwindow);
if(Target==Event.xbutton.subwindow)
    DispatchWindow=Target;
else
    DispatchWindow=PointerWindow;
```

Executing this code you have a good chance to guess the window to send the event to, *unless* the drop occurred in the root window. In this case the event is never sent, and a specific program routine must be called to handle this drop (if it is enabled at all). This concludes the drag protocol. Simple, huh? But you are advised to take a look at least at the function `DndHandleDragging` of the `Xt` interface to see how this was actually implemented.

3.3 Compatibility

If you want to receive drops from some application that uses the version 0 of the protocol, you will have to define two more atoms:

```
Atom OldDndProtocol,OldDndSelection;
OldDndProtocol=XInternAtom(display,"DndProtocol",False);
OldDndSelection=XInternAtom(display,"DndSelection",False);
```

Now to receive the drop from the older version you will have to test two things:

- If the client message type is `OldDndProtocol`.
- If the protocol version is 0. If the protocol version is not zero then you *must* ignore this message. A client message with protocol version greater than 0 and type `OldDndProtocol` is sent by some programs to enable older version programs to *receive* drops. But these programs send also a message with the correct protocol type, and you will receive this correct protocol version message.

If you want to send messages to applications compiled with older versions of the protocol, you must first make two copies of the data, one into the `DndSelection` and other into the `OldDndSelection`. Then you send *two* drop events to the application, one with type `DndProtocol` and other with type `OldDndProtocol`. *The protocol version field of the events must be greater than 0 in both events!* This will ensure that programs with the correct version will be able to discard the old fashioned message.

Chapter 4

Reference Guide

If you are curious enough to take a look at `DragAndDrop.h` you will see that there are non-documented functions in this file. As you may suspect, these functions are also not supported.

DndChangeCursor

Version 1.0

Syntax

void `DndChangeCursor`(int Type,
int width,
int height,
char *image,
char *mask,
int hot_x,
int hot_y);

Includes `OffiX/DragAndDrop.h`

Description Change the pointer cursor for a specified data type. The `image` and `mask` fields are arrays in the format of a bitmap (see `X` manuals). They must have the same dimensions. The fields `hot_x` and `hot_y` specifiy the hot spot coordinates for the new cursor.

Example

```
#include "my_file_cursor.xbm"
#include "my_file_mask.xbm"

int main()
{
    /* blah blah */

    DndChangeCursor(DndFile, my_file_cursor_width,
        my_file_cursor_height, my_file_cursor_bits,
        my_file_mask_bits, my_file_cursor_x_hot,
        my_file_cursor_y_hot);

    /* bloh bloh */
}
```

DndDataType

Version 1.0

Syntax

int `DndDataType`(XEvent *dropEvent);

Includes `OffiX/DragAndDrop.h`

Description Return the data type contained in the drop event, or the constant `DndNotDnd` if the event was not a DND drop event. For a description of the currently supported data types see section 2.3.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    long DataSize;
    char *Data;
    int type=DndDataType(event);
    if(type!=DndFile) return;
    DndGetData(&Data,&DataSize);
    LoadFile(Data);
}
```

DndDragButtons

Version 1.0

Syntax

unsigned int `DndDragButtons`(XEvent *dropEvent);

Includes `OffiX/DragAndDrop.h`

Description Return the buttons pressed for the drop event, in the format of the `xbutton.state` field of a button press/release of `X`.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int move;
    /* whatever */

    /* which buttons were pressed */
    button = DndDragButtons(event);
    move = button & Button1Mask ? True : False;
}
```

DndDropCoordinates

Version 1.0

Syntax

void `DndDropCoordinates`(Widget w,XEvent *dropEvent,int *x,int *y);

Includes `OffiX/DragAndDrop.h`

Description Return in `x` and `y` the coordinates of the drop point relative to the origin of the widget `w`. If the sender uses version 0 of the protocol, this function will return the current pointer position.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int x,y;
    if(!DndIsDropMessage(event)) return;
    DndDropCoordinates(w, event, &x, &y);
    fprintf(stderr,"Dropped at (%d,%d)\n",x,y);
}
```

DndDropRootCoordinates

Version 1.0

Syntax

void `DndDropRootCoordinates`(XEvent *dropEvent,int *x,int *y);

Includes `OffiX/DragAndDrop.h`

Description Return in `x` and `y` the coordinates of the drop point relative to the root window. If the sender has an older version of the protocol, this function will return the current pointer position.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int x,y;
    if(!DndIsDropMessage(event)) return;
    DndDropRootCoordinates(event, &x, &y);
    fprintf(stderr,"Dropped at (%d,%d)\n",x,y);
}
```

DndGetData

Version 1.0

Syntax

void `DndGetData`(unsigned char **data, unsigned long *size);

Includes `OffiX/DragAndDrop.h`

Description Retrieve the last drop data from the global DND buffer. The `size` field will contain the size of the `data` field, including NULL-terminating characters. For a more complete description of how to use this function see 2.4. This function does not allocate space for the data, so if your gonna stay with the data for a long time you must allocate space by yourself.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int DataType;
    unsigned long DataSize;
    char *Data;

    if(DndDataType(event)==DndNotDnd) return;
    DndGetData(& Data,& DataSize);

    /* ... data processing ... */
}
```

DndHandleDragging

Version 1.0

Syntax

int `DndHandleDragging`(Widget widget,XEvent *dragEvent);

Includes `OffiX/DragAndDrop.h`

Description Handles the drag action, i.e. grabs the pointer, waits for a button release and send the drop event accordingly. The function will return non-zero only if a drag really took place (for a drag to begin it is necessary to have a minimum 10-pixel pointer movement). You must set the drop data *before* calling this function.

Example

```
void myDragEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int DataType;
    unsigned long DataSize;
    char *Data;

    /* ... data initialization ... */

    DndSetData(DataType,Data,DataSize);
    DndHandleDragging(w,event);
}
```

DndInitialize

Version 1.0

Syntax

void `DndInitialize`(Widget toplevel);

Includes `OffiX/DragAndDrop.h`

Description Initialize the DND library internal structure. You must call this function after the creation of the top level widget, e.g. after `XtAppInitialize`. The argument is the top level widget of your application.

Example

```
int main()
{
    Widget shell;

    /* Application initializations */

    shell=XtAppInitialize(...);
    DndInitialize(shell);

    /* rest of the program */
}
```

DndIsDropMessage

Version 1.0

Syntax

int `DndIsDropMessage`(XEvent *event);

Includes `OffiX/DragAndDrop.h`

Description Return zero if and only if the event `event` is not a drop event from DND.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    if(!DndIsDropMessage(event)) return;

    /* now process the data in some way */
}
```

DndIsIcon

Version 1.0

Syntax

int `DndIsIcon`(Widget toplevel);

Includes `OffiX/DragAndDrop.h`

Description Returns non-zero if the top level widget `toplevel` is iconic. Performs a round-trip to the server.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    /* If we are iconified, de-iconify us */
    if(DndIsIcon(toplevel))
        XMapWindow(dpy,XtWindow(toplevel));
}
```

DndProtocolVersion

Version 1.0

Syntax

long `DndProtocolVersion`(XEvent *dropEvent);

Includes `OffiX/DragAndDrop.h`

Description Return the protocol version of the received event. If the event is not a DND event, then the function will return −1.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    if(DndProtocolVersion(event)<1)
        printf(stderr,"Protocol version too old!\n");
}
```

DndRegisterDragWidget

Version 1.0

Syntax

void `DndRegisterDragWidget`(Widget widget,
XtEventHandler handler,
XtPointer data);

Includes `OffiX/DragAndDrop.h`

Description Register a function to handle drags started by a specific widget of your application. This function is an `Xt` event handler (see section 2.1). This function will be called whenever a mouse button gets pressed while the pointer is on the widget. The `data` field can be safely replaced by NULL in the function call. For more information on handling drag events see 2.5.

Example

```
int main()
{
    Widget shell;

    /* Application initializations */

    shell=XtAppInitialize(...);
    DndInitialize(shell);

    toggle=XtCreateWidget(...);

    DndRegisterOtherDrop(myDropHandler);
    DndRegisterDragWidget(toggle,myToggleDragHandler,NULL);

    /* rest of the program */
}
```

DndRegisterDropWidget

Version 1.0

Syntax

void `DndRegisterDropWidget`(Widget widget,
XtEventHandler handler,
XtPointer data);

Includes `OffiX/DragAndDrop.h`

Description Register a function to handle drops on a specific widget of your application. This function is an `Xt` event handler (see section 2.1). This drop handler will have priority over handlers installed with `DndRegisterOtherDrop`. The `data` field can be safely replaced by NULL in the function call.

Example

```
int main()
{
    Widget shell;

    /* Application initializations */

    shell=XtAppInitialize(...);
    DndInitialize(shell);

    toggle=XtCreateWidget(...);

    DndRegisterOtherDrop(myDropHandler);
    DndRegisterDropWidget(toggle,myToggleDropHandler,NULL);

    /* rest of the program */
}
```

DndRegisterIconDrop

Version 1.0

Syntax

void `DndRegisterIconDrop`(XtEventHandler handler);

Includes `OffiX/DragAndDrop.h`

Description Register a function to handle drops on the icon window. This function will be called when your application receives a drop while it is iconified. This function is an `Xt` event handler (see section 2.1). Normally if you are able to receive drops on the icon window you are also able to receive drops anywhere within the application, so this function is rarely used. The function `DndRegisterOtherDrop` is used more often.

Example

```
int main()
{
    Widget shell;

    /* Application initializations */

    shell=XtAppInitialize(...);
    DndInitialize(shell);

    DndRegisterIconDrop(myIconDropHandler);

    /* rest of the program */
}
```

DndRegisterOtherDrop

Version 1.0

Syntax

void `DndRegisterOtherDrop`(XtEventHandler handler);

Includes `OffiX/DragAndDrop.h`

Description Register a function to handle drops on any non-registered window of your application. This function will be called when your application receives a drop on a non-registered widget. This function is an `Xt` event handler (see section 2.1). The effect of this function is to enable drops everywhere in your application.

Example

```
int main()
{
    Widget shell;

    /* Application initializations */

    shell=XtAppInitialize(...);
    DndInitialize(shell);

    DndRegisterOtherDrop(myDropHandler);

    /* rest of the program */
}
```

DndRegisterRootDrop

Version 1.0

Syntax

void `DndRegisterRootDrop`(XtEventHandler handler);

Includes `OffiX/DragAndDrop.h`

Description Register a function to handle drops on the root window. This function will be called when your application starts a drag and the drop occur in the root window. This function is an `Xt` event handler (see section 2.1).

Example

```
int main()
{
    Widget shell;

    /* Application initializations */

    shell=XtAppInitialize(...);
    DndInitialize(shell);

    DndRegisterRootDrop(myRootDropHandler);

    /* rest of the program */
}
```

DndSetData

Version 1.0

Syntax

void `DndSetData`(int type, unsigned char *data, unsigned long size);

Includes `OffiX/DragAndDrop.h`

Description Update the drop data in the global DND buffer. The `size` field must contain the size of the `data` field, including NULL-terminating characters. For a more complete description of how to use this function see 2.5.

Example

```
void myDragEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    int DataType;
    unsigned long DataSize;
    char *Data;

    DataType=DndFile;
    Data="/home/crusius/Makefile";
    DataSize=9L;

    DndSetData(DataType,Data,DataSize);
    DndHandleDragging(w,event);
}
```

DndSourceWindow

Version 1.0

Syntax

Window `DndSourceWindow`(XEvent *dropEvent);

Includes `OffiX/DragAndDrop.h`

Description Returns the `X` window of the sender. This window is *not* the main window of the sender client, but the *particular* window that started the drag. It can be used for test many things, including

- If the drop occurred at the same window where it started.
- If the drag-and-drop action occurred within the same application.

Example

```
void myDropEventHandler(Widget w,XtPointer data,
                        XEvent *event,Boolean *b)
{
    if(!DndIsDropMessage(event)) return;
    if(DndSenderWindow(event)==XtWindow(w))
        fprintf(stderr,"Dropped on same window!\n");
    return;
}
```

Index