– APPENDIX H –

THE DRAG & DROP PROTOCOL

Overview

The drag and drop protocol provides a simple method of data transmission between applications that support it. Because this protocol relies on the use of named pipes, the use of the drag and drop protocol is only possible under **MultiTOS**.

A drag and drop operation involves the user selecting a piece of program data (or perhaps several pieces) in the ‘originator’ application and dragging that piece of data with the mouse to the window of a ‘recipient’ application. This appendix will detail the drag and drop protocol from the perspective of the originator and the recipient.

You should note that during a drag and drop operation, neither application should lock the screen with **wind_update()**.

The Originator

When the user selects an object or group of objects, drags the mouse (and objects), and releases the mouse button outside one of your application window’s work areas, the operation is a candidate for a drag and drop operation.

When this action is initiated by the user, your application should call **wind_find()** to determine the window handle of the window at the drop location. From the window handle you can use **wind_get()** to determine the owner’s application identifier which will be needed to send an **AES** message to the application.

At this point you should use **Psignal()** to cause **SIGPIPE** (13) signals to be ignored and create a pipe named DRAGDROP.xx where ‘xx’ is a unique two character combination. The pipe created should have its ‘hidden’ attribute set. This causes reads to return **EOF** when the other end of the pipe is closed. To ensure your value is unique, try using the ASCII representation of your own application ID. If the **Fcreate()** fails, try a new combination until you find one that is available.

Now use **appl_write()** to send an **AES** message to the application whose window was targeted (the recipient) as follows:

WORD	Contents
0	**AP_DRAGDROP** (63)
1	Originator’s application id.
2	0
3	Window handle of the target.
4	Mouse X position at time of drop.
5	Mouse Y position at time of drop.
6	Keyboard shift status at time of drop.
7	2 character pipe ID packed into a **WORD** (this is the file extension of the created pipe).

The originator application should now use **Fselect()** to wait for either a write to the pipe or a timeout (3 to 4 seconds should be sufficient). If the call times out then the drag and drop operation failed and the pipe should be closed, otherwise, read one byte from the pipe which should be either **DD_OK** (0) or **DD_NAK** (1).

DD_OK means that the recipient wishes to continue the exchange. **DD_NAK** means that the user dropped the data on a window not prepared to accept data and that the pipe should be closed and the drag and drop operation aborted.

On receipt of a **DD_OK**, the originator should then read an additional 32 **BYTE**s from the pipe. These 32 **BYTE**s consist of eight 4 **BYTE** data type values that the recipient understands in order of preference. This list is not necessarily complete and the originator should not abort simply because it can't handle any of the listed data types. If less than eight data types are listed by the recipient the 32 bytes will be padded with zeros.

Data type values are four-byte ASCII values that represent data that might be exchanged. When these values are prefixed with a period, they represent data in a format that might be stored in a disk file. Examples of these are '.IMG', '.TXT', and '.GEM'. Some data types such as 'ARGS' or 'PATH' are not prefixed with a period because they represent special data.

The desktop sends an 'ARGS' drag and drop message to an application window when the user drags a group of file icons to an application window. The 'ARGS' data consists of a standard command line with the names of each file. 'ARGS' data should be translated for non-**TOS** file systems. Characters within single quotes should be interpreted as a single filename. Two single quotes in a row should be interpreted as a single quote.

After the originator has consulted the 32 byte list or preferred file types, it should construct its own structure consisting of the following data:

1. The type of data the originator has decided to send (4 ASCII bytes), ex: '.IMG'.
2. The length of data in bytes (**LONG**).
3. The data's name in ASCII format terminated by a **NULL** (this is a variable length field but should be brief as it will be used to label an icon which represents the data chunk), ex: "ASCII Text".
4. The filename the data is associated with in ASCII format terminated by a **NULL** (again, a variable length field), ex: "SAMPLE.TXT".

The originator should now write a **WORD** to the pipe signifying the length of the header and then the header itself. After doing so, the recipient will write a one byte reply indicating a return code from the following list:

Name	Value	Meaning
DD_OK	0	Ready to receive data. After receiving this message you should **Fwrite()** the actual data to the pipe and then **Fclose()** it to complete the operation.
DD_NAK	1	Abort the drag and drop. After receiving this message, close the pipe and abort the operation.
DD_EXT	2	The recipient cannot accept the format the data is in. You may either construct a new header and send it as before or close the pipe to abort the operation.
DD_LEN	3	The recipient cannot handle so much data. Either use a format which would cause less data to be sent or close the pipe to abort.
DD_TRASH	4	The data has been dropped on a trashcan. The pipe should be **Fclose()**'d and the data should be deleted from the originator application.
DD_PRINTER	5	The data has been dropped on a printer. The pipe should be **Fclose()**'d and the data should be printed.
DD_CLIPBOARD	6	The data has been dropped on a clipboard. The pipe should be **Fclose()**'d and the exchange should be treated like a 'Copy' clipboard operation.

The one exception to the above procedure involves the 'PATH' data type. If the recipient agrees to the 'PATH' data type by sending a **DD_OK**, the originator should *read* a path string (terminated by a **NULL** byte). The path string should be the complete pathname represented by the target window, ex: "C:\WORDPRO\FILES\". The size of the data, as specified in the header, specifies the maximum size of the string the recipient should write.

The Recipient

The drag and drop protocol begins for the recipient upon receipt of the **AP_DRAGDROP** message. When this message is received, the recipient should immediately open the pipe 'U:\PIPE\DRAGDROP.xx', where 'xx' is the two-byte ASCII identifier given in **WORD** 7 of the message, and write a **DD_OK** (0) to the pipe.

Next, as the recipient, you should construct a 32 byte structure consisting of eight 4 byte data names your application can receive. If your application recognizes less than eight types of data pad the 32 bytes with zeros. After this structure is constructed, write it to the pipe.

Now you should read a **WORD** from the pipe which will indicate the size of the message header which should be read immediately after. The message header consists of a four byte ASCII data type, a **LONG** indicating the size of the data, a **NULL** terminated string of variable size which identifies the data (or simply **NULL** if none), and a **NULL** terminated filename (or **NULL** if none).

After decoding the message header you should respond with one of the one-byte response codes as listed in the previous table. If the recipient cannot process the data type sent, it should send **DD_EXT** and wait for reception of another header (preceded again by a **WORD** headed size). If the originator cannot supply any more data types you will receive a 0 byte return from the **Fread()** call and you should **Fclose()** the pipe and abort.

If the data type is acceptable, respond with **DD_OK**, read the number of data bytes as indicated in the header to receive the actual data, and then close the pipe.

A special case arises if the header specifies ‘PATH’ as a data type. In this case you should send a **DD_OK** message (if appropriate) and write the pathname associated with the target window (you can write as many bytes as is specified in the message header data length).