Appendix C Standard DTM Routines

Overview

This appendix lists some standard DTM routines described by DTM's creator, Jeff Terstriep. All DTM documentation is available via FTP. Refer to Appendix C, "Obtaining NCSA Software," for those instructions.

DTMmakeInPort

```
int DTMmakeInPort( portname )
char *portname;
```

`DTMmakeInPort` creates an input port. *Portname* is pointer to a string with the format 'hostname:port'. 'Hostname' is optional and will always be replaced with the local host's name. *Portname* represent the address where the system will listen for incoming messages. If *portname* is ':0' then the system will assign the TCP port number, the value can be retrieved with `DTMgetPortAddr` (see below).

If `DTMmakeInPort` suceeds, it returns a *portid*. The *portid* is a small integer used to refer to the port in all subsequent DTM calls. If there is any problem `DTMmakeInPort` will return `DTMERROR`.

Possible errors are returned by `DTMmakeInPort`:

DTMNOPORT	No more open DTM ports.
DTMMEM	Insufficient memory for port.
DTMHUH	Illegal port name.

DTMmakeOutPort

```
int DTMmakeOutPort( portname )
char *portname;
```

`DTMmakeOutPort` creates an output port. *Portname* is pointer to a string with the format 'hostname:port'. *Portname* represents the address where outgoing messages will be sent. Therefore, 'hostname' is any legal host name or IP address. 'Port' is a TCP port number where an application is listening, possably through the use of `DTMmakeInPort.`

If `DTMmakeOutPort` suceeds, it returns a *portid*. The *portid* is a small integer used to refer to the port in all subsequent DTM calls. If there is any problem `DTMmakeOutPort` will return `DTMERROR.`

Possible errors are returned by `DTMmakeOutPort`:

`DTMNOPORT`	No more open DTM ports.
`DTMMEM`	Insufficient memory for port.
`DTMHUH`	Illegal port name.

DTMgetPortAddr

```
int DTMgetPortAddr( portid, address, size )
int  portid
char *address;
int  size;
```

`DTMgetPortAddr` returns the IP address of DTM port. This is typically used in conjunction with `DTMmakeInPort(":0")` to retrieve the TCP port number and report it to connecting programs.

Portid is value returned on a previous call to `DTMmakeInPort`. *Address* is a buffer where the address in the form 'hostname:port' will be stored. *Size* is the size of the *Address* buffer.

Possible errors are returned by `DTMgetPortAddr`:

`DTMPORTINIT`	invalid value for *portid*.

DTMavailRead

```
int DTMavailRead( portid )
int  portid;
```

`DTMavailRead` performs a non-blocking check for a message on the input port *portid*. `DTMavailRead` returns TRUE (1) if a message is available and FALSE (0) if not. `DTMavailRead` will return `DTMERROR` if a problem is encountered. Since `DTMERROR` also represents a TRUE value, an application can check for the possibility of an error by examining `DTMerrno`, for a non-zero state, after the call.

Possible errors are returned by `DTMavailRead`:

`DTMPORTINIT`	invalid value for *portid*.
`DTMSOCK`	problem creating connection.

DTMavailWrite

```
int DTMavailWrite( portid )
int  portid;
```

`DTMavailWrite` performs a non-blocking check, on the output port *portid*, to determine if the receiving program has processed the previous message. `DTMavailWrite` returns TRUE (1) if a message is available and FALSE (0) if not. `DTMavailWrite` will return

`DTMERROR` if a problem is encountered. Since `DTMERROR` also represents a TRUE value, an application can check for the possibility of an error by examining `DTMerrno`, for a non-zero state, after the call.

Possible errors are returned by `DTMavailWrite`:

`DTMPORTINIT`	invalid value for *portid*.
`DTMSOCK`	problem creating connection.

DTMbeginRead

```
int DTMbeginRead( portid, header, size )
int  portid;
char *header;
int  size;
```

`DTMbeginRead` receives a message from the input port *portid*. The message header is placed in the buffer *header*. If no message is currently available, this call will block. A non-blocking check for a pending message may be performed with DTMavailRead (see above).

Size indicates the size of the buffer allocated to hold the incoming header. `DTM_MAX_HEADER` is defined to be the largest legal header length and may be used to allocate the header buffer. If the incoming header is larger than the header buffer, `DTMbeginRead` will fill the header buffer, discard the remaning header and return `DTMERROR`. In this case `DTMerrno` will be set to `DTMHEADER`.

Possible errors are returned by `DTMbeginRead`:

`DTMPORTINIT`	invalid value for *portid*.
`DTMSOCK`	problem creating connection.
`DTMREAD`	problem reading from connection.
`DTMHEADER`	incoming header exceeds buffer size.

DTMbeginWrite

```
int DTMbeginWrite( portid, header, size )
int  portid;
char *header;
int  size;
```

`DTMbeginWrite` writes the header of a message to the output port *portid*. If the previous message has not been received this call will block. A non-blocking check to determine if the previous message has been received is available with `DTMavailWrite` (see above).

Header is a buffer containing the header of the message to be written. *Size* is the length of the header, it may be calculated with `DTMheaderLength(`*`header`*`).`

Possible error condition from `DTMbeginWrite`:

`DTMPORTINIT`	Invalid value for *portid*.
`DTMSOCK`	Problem creating connection.
`DTMTIMEOUT`	Time-out waiting for receiver.

DTMWRITE	Error writing header.

DTMrecvDataset

```
int DTMrecvDataset( portid, buffer, num_elements, type )
int  portid;
char *buffer;
int  num_elements;
DTMTYPE   type;
```

`DTMrecvDataset` reads the data section of a message from the input port *portid*. This call is optional, if it is used it must be preceeded by a call to `DTMbeginRead`. `DTMrecvDataset` will attempt to fill the buffer with number of elements of the specified type, automatic type conversion will be performed where necessary. *Buffer* is assumed to be large enough to hold the amount of data requested.

In the absence of errors, `DTMrecvDataset` returns the number of elements actually read. The process may call `DTMrecvDataset` as often as required to receive the message in its entirety, the value returned from `DTMrecvDataset` will equal 0 at the end of the message.

Possible error conditions from `DTMrecvDataset`:

`DTMCALL`	`DTMbeginRead` must preceed this call.
`DTMREAD`	Error reading message.

DTMsendDataset

```
int DTMsendDataset( portid, buffer, num_elements, type )
int  portid;
char *buffer;
int  num_elements;
DTMTYPE   type;
```

`DTMsendDataset` writes the data section of a message to the output port *portid*. This call is optional, if it is used it must be preceeded by a call to `DTMbeginWrite`. `DTMsendDataset` will write the number of elements of the specified type from the buffer, automatic type conversion will be performed where necessary. `DTMsendDataset` may be called as often as necessary to complete the message.

Possible error conditions from `DTMsendDataset`:

`DTMCALL`	`DTMbeginWrite` must preceed call.
`DTMWRITE`	Error writing message.

DTMendRead

```
int DTMendRead( portid )
int  portid;
```

DTMendRead marks the end of the current message and prepares for the next message on the input port *portid*. Any data remaining in the message is discarded. There must be a matching

`DTMendRead` for every call to `DTMbeginRead`.

Possible error conditions from `DTMendRead`:

`DTMCALL`	`DTMbeginRead` must preceed call.

DTMendWrite

```
int DTMendWrite( portid )
int  portid;
```

`DTMendWrite` marks the end of the current message. There must be a matching `DTMendWrite` for every call to `DTMbeginWrite`.

Possible error conditions from `DTMendWrite`:

`DTMCALL`	`DTMbeginWrite` must preceed call.

DTMdestoryPort

```
int DTMdestoryPort( portid )
int  portid;
```

`DTMdestroyPort` closes all connections associated with the port *portid* and frees the entry in the port table. This call, although optional, is recommended since it may assist connected processes in proceeding correctly.

Possible error conditions from `DTMdestroyPort`:

`DTMPORTINIT`	Invalid *portid*.