X Transport Interface

Designed by Stuart Anderson (NCR) with help from Ralph Mor (X Consortium)

Draft Version 0.6

NOTE: This documentation does not completely match the implementation in R6 (as a result of some late changes made in the code). Specifically, support was added for font server cloning, and conditional compliation was introduced for client vs. server code.

1. Purposes and Goals

The X Transport Interface is intended to combine all system and transport specific code into a single place in the source tree. This API should be used by all libraries, clients and servers of the X Window System. Use of this API should allow the addition of new types of transports and support for new platforms without making any changes to the source except in the X Transport Interface code.

This interface should solve the problem of multiple #ifdef TRANSPORT and #ifdef PLATFORM statements scattered throughout the source tree.

This interface should provide enough functionality to support all types of protocols, including connection oriented protocols such as X11 and FS, and connection-less oriented protocols such as XDMCP.

2. Overview of the interface

The interface provides an API for use by applications. The functions in this API perform work that is common to all transports and systems, such as parsing an address into a host and port number. The functions in this API call transport specific functions that are contained in a table whose contents are defined at compile time. This table contains an entry for each type of transport. Each entry is a record containing mostly pointers to function that implements the interface for the given transport.

This API does not provide an abstraction for select() or poll(). These function are themselves transport independent, so an additional interface is not needed for these functions. It is also unclear how such an interface would affect performance.

3. Definition of Address Specification Format

Addresses are specified in the following syntax,

protocol/host:port

where *protocol* specifies a protocol family or an alias for a protocol family. A definition of common protocol families is given in a later section.

The *host* part specifies the name of a host or other transport dependent entity that could be interpreted as a Network Service Access Point (NSAP).

The *port* part specifies the name of a Transport Service Access Point (TSAP). The format of the TSAP is defined by the underlying transport implementation, but it is represented using a string format when it is part of an address.

4. Internal Data Structures

There are two major data structures associated with the transport independent portion of this interface. Additional data structures may be used internally by each transport.

4.1 Xtransport

Each transport supported has an entry in the transport table. The transport table is an array of *Xtransport* records. Each record contains all the entry points for a single transport. This record is defined as:

```
typedef struct _Xtransport {

    char *TransName;
    int  flags;

    XtransConnInfo (*OpenCOTSClient)(
            struct _Xtransport *,       /* transport */
            char *,                     /* protocol */
            char *,                     /* host */
            char *                      /* port */
    );

    XtransConnInfo (*OpenCOTSServer)(
            struct _Xtransport *,       /* transport */
            char *,                     /* protocol */
            char *,                     /* host */
            char *                      /* port */
    );

    XtransConnInfo (*OpenCLTSClient)(
            struct _Xtransport *,       /* transport */
            char *,                     /* protocol */
            char *,                     /* host */
            char *                      /* port */
    );

    XtransConnInfo (*OpenCLTSServer)(
            struct _Xtransport *,       /* transport */
            char *,                     /* protocol */
            char *,                     /* host */
            char *                      /* port */
    );

    int     (*SetOption)(
            XtransConnInfo,             /* connection */
            int,                        /* option */
            int                         /* arg */
    );

    int     (*CreateListener)(
            XtransConnInfo,             /* connection */
            char *                      /* port */
    );

    int     (*ResetListener)(
            XtransConnInfo              /* connection */
    );

    XtransConnInfo (*Accept)(
            XtransConnInfo              /* connection */
    );

    int     (*Connect)(
            XtransConnInfo,             /* connection */
            char *,                     /* host */
```

```
        char *                      /* port */
    );

    int     (*BytesReadable)(
        XtransConnInfo,             /* connection */
        BytesReadable_t *           /* pend */
    );

    int     (*Read)(
        XtransConnInfo,             /* connection */
        char *,                     /* buf */
        int                         /* size */
    );

    int     (*Write)(
        XtransConnInfo,             /* connection */
        char *,                     /* buf */
        int                         /* size */
    );

    int     (*Readv)(
        XtransConnInfo,             /* connection */
        struct iovec *,             /* buf */
        int                         /* size */
    );

    int     (*Writev)(
        XtransConnInfo,             /* connection */
        struct iovec *,             /* buf */
        int                         /* size */
    );

    int     (*Disconnect)(
        XtransConnInfo              /* connection */
    );

    int     (*Close)(
        XtransConnInfo              /* connection */
    );

} Xtransport;
```

The *flags* field can contain an OR of the following masks:

TRANS_ALIAS: indicates that this record is providing an alias, and should not be used to create a listner.

TRANS_LOCAL: indicates that this is a LOCALCONN transport.

4.2 XtransConnInfo

Each connection will have an opaque *XtransConnInfo* transport connection object allocated for it. This record contains information specific to the connection. The record is defined as:

```
typedef struct _XtransConnInfo *XtransConnInfo;

struct _XtransConnInfo {
    struct _Xtransport     *transptr;
    char    *priv;
    int                flags;
    int                fd;
    int                family;
    char    *addr;
    int                addrlen;
    char    *peeraddr;
    int                peeraddrlen;
};
```

5. Exposed Transport Independent API

This API is included in each library and server that uses it. The API may be used by the library, but it is not added to the public API for that library. This interface is simply an implementation facilitator. This API contains a low level set of core primitives, and a few utility functions that are built on top of the primitives. The utility functions exist to provide a more familiar interface that can be used to port existing code.

A macro is defined in Xtrans.h for TRANS(func) that creates a unique function name depending on where the code is compiled. For example, when built for Xlib, TRANS(OpenCOTSClient) becomes _X11TransOpenCOTSClient.

All failures are considered fatal, and the connection should be closed and re-established if desired. In most cases, however, the value of errno will be available for debugging purposes.

5.1 Core Interface API

- XtransConnInfo TRANS(OpenCOTSClient)(char *address)

  This function creates a Connection-Oriented Transport that is suitable for use by a client. The parameter *address* contains the full address of the server to which this endpoint will be connected. This functions returns an opaque transport connection object on success, or NULL on failure.

- XtransConnInfo TRANS(OpenCOTSServer)(char *address)

  This function creates a Connection-Oriented Transport that is suitable for use by a server. The parameter *address* contains the full address to which this server will be bound. This functions returns an opaque transport connection object on success, or NULL on failure.

- XtransConnInfo TRANS(OpenCLTSClient)(char *address)

  This function creates a Connection-Less Transport that is suitable for use by a client. The parameter *address* contains the full address of the server to which this endpoint will be connected. This functions returns an opaque transport connection object on success, or NULL on failure.

- XtransConnInfo TRANS(OpenCLTSServer)(char *address)

  This function creates a Connection-Less Transport that is suitable for use by a server. The parameter *address* contains the full address to which this server will be bound. This functions returns an opaque transport connection object on success, or NULL on failure.

- int TRANS(SetOption)(XtransConnInfo connection, int option, int arg)

  This function sets transport options, similar to the way setsockopt() and ioctl() work. The parameter *connection* is an endpoint that was obtained from _XTransOpen*() functions. The parameter *option* contains the option that will be set. The actual values for *option* are defined in a later section. The parameter *arg* can be used to pass in an additional value that may be required by some options. This function return 0 on success and -1 on failure.

Note: Based on current usage, the complimentary function TRANS(GetOption)() is not necessary.

- int TRANS(CreateListener)(XtransConnInfo connection, char *port)

  This function sets up the server endpoint for listening. The parameter *connection* is an endpoint that was obtained from TRANS(OpenCOTSServer)() or TRANS(OpenCLTSServer)(). The parameter *port* specifies the port to which this endpoint should be bound for listening. If *port* is NULL, then the transport may attempt to allocate any available TSAP for this connection. If the transport cannot support this, then this function will return a failure. This function return 0 on success and -1 on failure.

- int TRANS(ResetListener)(XtransConnInfo connection)

  When a server is restarted, certain listen ports may need to be reset. For example, unix domain needs to check that the file used for communication has not been deleted. If it has, it must be recreated. The parameter *connection* is an opened and bound endpoint that was obtained from TRANS(OpenCOTSServer)() and passed to TRANS(CreateListener)(). This function will return one of the following values: TRANS_RESET_NOOP, TRANS_RESET_NEW_FD, or TRANS_RESET_FAILURE.

- XtransConnInfo TRANS(Accept)(XtransConnInfo connection)

  Once a connection indication is received, this function can be called to accept the connection. The parameter *connection* is an opened and bound endpoint that was obtained from TRANS(OpenCOTSServer)() and passed to TRANS(CreateListener)(). This function will return a new opaque transport connection object upon success, NULL otherwise.

- int TRANS(Connect)(XtransConnInfo connection, char *address)

  This function creates a connection to a server. The parameter *connection* is an endpoint that was obtained from TRANS(OpenCOTSClient)(). The parameters *address* specify the TSAP to which this endpoint should connect. If the protocol is included in the address, it will be ignored. This function return 0 on success and -1 on failure.

- int TRANS(BytesReadable)(XtransConnInfo connection, BytesReadable_t *pend);

  This function provides the same functionality as the BytesReadable macro.

- int TRANS(Read)(XtransConnInfo connection, char *buf, int size)

  This function will return the number of bytes requested on a COTS connection, and will return the minimum of the number bytes requested or the size of the incoming packet on a CLTS connection.

- int TRANS(Write)(XtransConnInfo connection, char *buf, int size)

  This function will write the requested number of bytes on a COTS connection, and will send a packet of the requested size on a CLTS connection.

- int TRANS(Readv)(XtransConnInfo connection, struct iovec *buf, int size)

  Similar to TRANS(Read)().

- int TRANS(Writev)(XtransConnInfo connection, struct iovec *buf, int size)

  Similar to TRANS(Write)().

- int TRANS(Disconnect)(XtransConnInfo connection)

  This function is used when an orderly disconnect is desired. This function breaks the connection on the transport. It is similar to the socket function shutdown().

- int TRANS(Close)(XtransConnInfo connection)

  This function closes the transport, unbinds it, and frees all resources that was associated with the transport. If a TRANS(Disconnect) call was not made on the connection, a disorderly disconnect may occur.

- int TRANS(IsLocal)(XtransConnInfo connection)

  Returns TRUE if it is a local transport.

- int TRANS(GetMyAddr)(XtransConnInfo connection,
  int *familyp, int *addrlenp, Xtransaddr **addrp)

  This function is similar to getsockname(). This function will allocate space for the address, so it must be freed by the caller. Not all transports will have a valid address until a connection is established. This function should not be used until the connection is established with Connect() or Accept().

- int TRANS(GetPeerAddr)(XtransConnInfo connection,
  int *familyp, int *addrlenp, Xtransaddr **addrp)

  This function is similar to getpeername(). This function will allocate space for the address, so it must be freed by the caller. Not all transports will have a valid address until a connection is established. This function should not be used until the connection is established with Connect() or Accept().

- int TRANS(GetConnectionNumber)(XtransConnInfo connection)

  Returns the file descriptor associated with this transport.

- int TRANS(MakeAllCOTSServerListeners)(
  char *port, int *partial_ret, int *count_ret, XtransConnInfo **connections_ret)

  This function should be used by most servers. It will try to establish a COTS server endpoint for each transport listed in the transport table. *partial_ret* will be set to True if only a partial network could be created. *count_ret* is the number of transports returns, and *connections_ret* is the list of transports.

- int TRANS(MakeAllCLTSServerListeners)(
  char *port, int *partial_ret, int *count_ret, XtransConnInfo **connections_ret)

  This function should be used by most servers. It will try to establish a CLTS server endpoint for each transport listed in the transport table. *partial_ret* will be set to True if only a partial network could be created. *count_ret* is the number of transports returns, and *connections_ret* is the list of transports.

5.2 Utility API

This section describes a few useful functions that have been implemented on top of the Core Interface API. These functions are being provided as a convenience.

- int TRANS(ConvertAddress)(int *familyp, int *addrlenp, Xtransaddr *addrp)

  This function converts a sockaddr based address to an X authorization based address (ie AF_INET, AF_UNIX to the X protocol definition (ie FamilyInternet, FamilyLocal)).

6. Transport Option Definition

The following options are defined for the TRANS(SetOption)() function. If an OS or transport does not support any of these options, then it will silently ignore the option.

- TRANS_NONBLOCKING

  This option controls the blocking mode of the connection. If the argument is set to 1, then the connection will be set to blocking. If the argument is set to 0, then the connection will be set to non-blocking.

- TRANS_CLOSEONEXEC

  This option determines what will happen to the connection when an exec is encountered. If the argument is set to 1, then the connection will be closed when an exec occurs. If the argument is set to 0, then the connection will not be closed when an exec occurs.

7. Hidden Transport Dependent API

The hidden transport dependent functions are placed in the Xtransport record. These function are similar to the Exposed Transport Independent API, but some of the parameters and return values are slightly different. Stuff like the #ifdef SUNSYSV should be handled inside these functions.

- XtransConnInfo *OpenCOTSClient (
  struct _Xtransport *thistrans, char *protocol, char *host, char *port)

  This function creates a Connection-Oriented Transport. The parameter *thistrans* points to an Xtransport entry in the transport table. The parameters *protocol*, *host*, and *port* point to strings containing the corresponding parts of the address that was passed into TRANS(OpenCOTSClient)().

  This function must allocate and initialize the contents of the XtransConnInfo structure that is returned by this function. This function will open the transport, and bind it into the transport namespace if applicable. The local address portion of the XtransConnInfo structure will also be filled in by this function.

- XtransConnInfo *OpenCOTSServer (
  struct _Xtransport *thistrans, char *protocol, char *host, char *port)

  This function creates a Connection-Oriented Transport. The parameter *thistrans* points to an Xtransport entry in the transport table. The parameters *protocol*, *host*, and *port* point to strings containing the corresponding parts of the address that was passed into TRANS(OpenCOTSClient)().

  This function must allocate and initialize the contents of the XtransConnInfo structure that is returned by this function. This function will open the transport.

- XtransConnInfo *OpenCLTSClient (
  struct _Xtransport *thistrans, char *protocol, char *host, char *port)

  This function creates a Connection-Less Transport. The parameter *thistrans* points to an Xtransport entry in the transport table. The parameters *protocol*, *host*, and *port* point to strings containing the corresponding parts of the address that was passed into TRANS(OpenCOTSClient)().

  This function must allocate and initialize the contents of the XtransConnInfo structure that is returned by this function. This function will open the transport, and bind it into the transport namespace if applicable. The local address portion of the XtransConnInfo structure will also be filled in by this function.

- XtransConnInfo *OpenCLTSServer (
  struct _Xtransport *thistrans, char *protocol, char *host, char *port)

  This function creates a Connection-Less Transport. The parameter *thistrans* points to an Xtransport entry in the transport table. The parameters *protocol*, *host*, and *port* point to strings containing the corresponding parts of the address that was passed into TRANS(OpenCOTSClient)().

  This function must allocate and initialize the contents of the XtransConnInfo structure that is returned by this function. This function will open the transport.

- int SetOption (struct _Xtransport *thistrans, int option, int arg)

  This function provides a transport dependent way of implementing the options defined by the X Transport Interface. In the current prototype, this function is not being used, because all of the option defined so far, are transport independent. This function will have to be used if a radically different transport type is added, or a transport dependent option is defined.

- int CreateListener (struct _Xtransport *thistrans, char *port)

  This function takes a transport endpoint opened for a server, and sets it up to listen for incoming connection requests. The parameter *port* should contain the port portion of the address that was passed to the Open function.

This function will bind the transport into the transport name space if applicable, and fill in the local address portion of the XtransConnInfo structure. The transport endpoint will then be set to listen for incoming connection requests.

- int ResetListener (struct _Xtransport *thistrans)

  This function resets the transport for listening.

- XtransConnInfo Accept(struct _Xtransport *thistrans)

  This function creates a new transport endpoint as a result of an incoming connection request. The parameter *thistrans* is the endpoint that was opened for listening by the server. The new endpoint is opened and bound into the transport's namespace. A XtransConnInfo structure describing the new endpoint is returned from this function

- int Connect(struct _Xtransport *thistrans, char *host, char *port)

  This function establishes a connection to a server. The parameters *host* and *port* describe the server to which the connection should be established. The connection will be established so that Read() and Write() call can be made.

- int BytesReadable(struct _Xtransport *thistrans, BytesReadable_t *pend)

  This function replaces the BytesReadable() macro. This allows each transport to have it's own mechanism for determining how much data is ready to be read.

- int Read(struct _Xtransport *thistrans, char *buf, int size)

  This function reads *size* bytes into *buf* from the connection.

- int Write(struct _Xtransport *thistrans, char *buf, int size)

  This function writes *size* bytes from *buf* to the connection.

- int Readv(struct _Xtransport *thistrans, struct iovec *buf, int size)

  This function performs a readv() on the connection.

- int Writev(struct _Xtransport *thistrans, struct iovec *buf, int size)

  This function performs a writev() on the connection.

- int Disconnect(struct _Xtransport *thistrans)

  This function initiates an orderly shutdown of a connection. If a transport does not distinguish between orderly and disorderly disconnects, then a call to this function will have no affect.

- int Close(struct _Xtransport *thistrans)

  This function will break the connection, and close the endpoint.

8. Configuration

The implementation of each transport can be platform specific. It is expected that existing connection types such as TCPCONN, UNIXCONN, LOCALCONN and STREAMSCONN will be replaced with flags for each possible transport type.

Below are the flags that can be set in *ConnectionFlags* in the vendor.cf or site.def config files.

TCPCONN	Enables the INET Domain Socket based transport
UNIXCONN	Enables the UNIX Domain Sokcet based transport
STREAMSCONN	Enables the TLI based transports
LOCALCONN	Enables the SYSV Local connection transports
DNETCONN	Enables the DECnet transports

9. Transport Specific Definitions

Protocol Family	Address Component		
	protocol	**host**	**port**
Internet	inet tcp udp	name of an internet addressable host	string containing the name of a service or a valid port number. Example: "xserver0", "7100"
DECnet	decnet	name of a DECnet addressable host	string containing the complete name of the object. Example: "X$X0"
NETware	ipx	name of a NETware addressable host	Not sure of the specifics yet.
OSI	osi	name of an OSI addressable host	Not sure of the specifics yet.
Local	local pts named sco isc	(ignored)	String containing the port name, ie "xserver0", "fontserver0".

10. Implementation Notes

This section refers to the prototype implementation that is being developed concurrently with this document. This prototype has been able to flush out many details and problems as the specification was being developed.

All of the source code for this interface is located in xc/lib/xtrans.

All functions names in the source are of the format TRANS(func)(). The TRANS() macro is defined as

```
#if (__STDC__ && !defined(UNIXCPP)) || defined(ANSICPP)
#define TRANS(func) _PROTOCOLTrans##func
#else
#define TRANS(func) _PROTOCOLTrans/**/func
#endif
```

PROTOCOL will be uniquely defined in each directory where this code is compiled. PROTOCOL will be defined to be the name of the protocol that is implemented by the library or server, such as X11, FS, and ICE.

All libraries and servers that use the X Transport Interface should have a new file called transport.c. This file will include the transports based on the configuration flags *ConnectionFlags*. Below is an example transport.c.

```
#include "Xtransint.h"

#ifdef DNETCONN
#include "Xtransdnet.c"
#endif
#ifdef LOCALCONN
#include "Xtranslocal.c"
#endif
#ifdef TCPCONN
#include "Xtranssock.c"
#endif
#ifdef STREAMSCONN
#include "Xtranstli.c"
#endif
#include "Xtrans.c"
#include "Xtransutil.c"
```

The source files for this interface are listed below.

Xtrans.h	Function prototypes and defines for the Transport Independent API.
Xtransint.h	Used by the interface implementation only. Contains the internal data structures.
Xtranssock.c	Socket implementation of the Transport Dependent API.
Xtranstli.c	TLI implementation of the Transport Dependent API.
Xtransdnet.c	DECnet implementation of the Transport Dependent API.
Xtranslocal.c	Implementation of the Transport Dependent API for SYSV Local connections.
Xtrans.c	Exposed Transport Independent API Functions.
Xtransutil.c	Collection of Utility functions that use the X Transport Interface.

The file *Xtransint.h* contains much of the transport related code that previously in Xlibint.h and Xlibnet.h. This will make the definitions available for all transport users. This should also obsolete the equivilent code in other libraries.