The XIM Transport Specification

Revision 0.1

X Version 11, Release 6.4

Takashi Fujiwara
FUJITSU LIMITED

ABSTRACT

This specification describes the transport layer interfaces between Xlib and IM Server, which makes various channels usable such as X protocol or, TCP/IP, DECnet and etc.

-- --

Copyright © 1994 by FUJITSU LIMITED

Permission to use, copy, modify, and distribute this documentation for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. Fujitsu makes no representations about the suitability for any purpose of the information in this document. This documentation is provided as is without express or implied warranty.

Copyright © 1994 X Consortium

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ''Software''), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ''AS IS'', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.

X Window System is a trademark of X Consortium, Inc.

1. Introduction

The Xlib XIM implementation is layered into three functions, a protocol layer, an interface layer and a transport layer. The purpose of this layering is to make the protocol independent of transport implementation. Each function of these layers are:

The protocol layer
: implements overall function of XIM and calls the interface layer functions when it needs to communicate to IM Server.

The interface layer
: separates the implementation of the transport layer from the protocol layer, in other words, it provides implementation independent hook for the transport layer functions.

The transport layer
: handles actual data communication with IM Server. It is done by a set of several functions named transporters.

This specification describes the interface layer and the transport layer, which makes various communication channels usable such as X protocol or, TCP/IP, DECnet, STREAM, etc., and provides the information needed for adding another new transport layer. In addition, sample implementations for the transporter using the X connection is described in section 4.

2. Initialization

2.1. Registering structure to initialize

The structure typed as TransportSW contains the list of the transport layer the specific implementations supports.

```
typedef struct {
    char *transport_name;
    Bool (*config);
} TransportSW;
```

transport_name name of transport(*1)

config initial configuration function

A sample entry for the Xlib supporting transporters is shown below:

```
TransportSW _XimTransportRec[] = {
/*      char *:
 *      transport_name,            Bool (*config)()
 */
        "X",                       _XimXConf,
        "tcp",                     _XimTransConf,
        "local",                   _XimTransConf,
        "decnet",                  _XimTransConf,
        "streams",                 _XimTransConf,
        (char *)NULL,              (Bool (*)())NULL,
};
```

(*1) Refer to "The Input Method Protocol: Appendix B"

2.2. Initialization function

The following function will be called once when Xlib configures the transporter functions.

Bool (*config)(*im*, *transport_data*)
  XIM *im*;
  char **transport_data*;

im	Specifies XIM structure address.
transport_data	Specifies the data specific to the transporter, in IM Server address. (*1)

This function must setup the transporter function pointers.

The actual *config* function will be chosen by IM Server at the pre-connection time, matching by the *transport_name* specified in the **_XimTransportRec** array; The specific members of XimProto structure listed below must be initialized so that point they appropriate transporter functions.

If the specified transporter has been configured successfully, this function returns True. There is no Alternative Entry for config function itself.

The structure XimProto contains the following function pointers:

```
Bool (*connect)();              /* Open connection */
Bool (*shutdown)();             /* Close connection */
Bool (*write)();                /* Write data */
Bool (*read)();                 /* Read data */
Bool (*flush)();                /* Flush data buffer */
Bool (*register_dispatcher)();  /* Register asynchronous data handler */
Bool (*call_dispatcher)();      /* Call dispatcher */
```

These functions are called when Xlib needs to communicate the IM Server. These functions must process the appropriate procedure described below.

3. The interface/transport layer functions

Following functions are used for the transport interface.

Table 3-1; The Transport Layer Functions.

Alternative Entry (Interface Layer)	XimProto member (Transport Layer)	Section
_XimConnect	connect	3.1
_XimShutdown	shutdown	3.2
_XimWrite	write	3.3
_XimRead	read	3.4
_XimFlush	flush	3.5
_XimRegisterDispatcher	register_dispatcher	3.6
_XimCallDispatcher	call_dispatcher	3.7

The Protocol layer calls the above functions using the Alternative Entry in the left column. The transport implementation defines XimProto member function in the right column. The Alternative Entry is provided so as to make easier to implement the Protocol Layer.

(*1) Refer to "The Input Method Protocol: Appendix B"

3.1. Opening connection

When **XOpenIM** is called, the following function is called to connect with the IM Server.

Bool (*connect)(*im*)
XIM *im*;

im Specifies XIM structure address.

This function must establishes the connection to the IM Server. If the connection is established successfully, this function returns True. The Alternative Entry for this function is:

Bool _XimConnect(*im*)
XIM *im*;

im Specifies XIM structure address.

3.2. Closing connection

When **XCloseIM** is called, the following function is called to disconnect the connection with the IM Server. The Alternative Entry for this function is:

Bool (*shutdown)(*im*)
XIM *im*;

im Specifies XIM structure address.

This function must close connection with the IM Server. If the connection is closed successfully, this function returns True. The Alternative Entry for this function is:

Bool _XimShutdown(*im*)
XIM *im*;

im Specifies XIM structure address.

3.3. Writing data

The following function is called, when Xlib needs to write data to the IM Server.

Bool (*write)(*im*, *len*, *data*)
XIM *im*;
INT16 *len*;
XPointer *data*;

im Specifies XIM structure address.

len Specifies the length of writing data.

data Specifies the writing data.

This function writes the *data* to the IM Server, regardless of the contents. The number of bytes is passed to *len*. The writing data is passed to *data*. If data is sent successfully, the function returns True. Refer to "The Input Method Protocol" for the contents of the writing data. The Alternative Entry for this function is:

Bool _XimWrite(*im*, *len*, *data*)
XIM *im*;
INT16 *len*;
XPointer *data*;

im Specifies XIM structure address.

len	Specifies the length of writing data.
data	Specifies the writing data.

3.4. Reading data

The following function is called when Xlib waits for response from IM server synchronously.

```
Bool (*read)(im, read_buf, buf_len, ret_len)
    XIM im;
    XPointer read_buf;
    int buf_len;
    int *ret_len;
```

im	Specifies XIM structure address.
read_buf	Specifies the buffer to store data.
buf_len	Specifies the size of the *buffer*
ret_len	Specifies the length of stored data.

This function stores the read data in *read_buf*, which size is specified as *buf_len*. The size of data is set to *ret_len*. This function return True, if the data is read normally or reading data is completed.

The Alternative Entry for this function is:

```
Bool _XimRead(im, ret_len, buf, buf_len, predicate, predicate_arg)
    XIM im;
    INT16 *ret_len;
    XPointer buf;
    int buf_len;
    Bool (*predicate)();
    XPointer predicate_arg;
```

im	Specifies XIM structure address.
ret_len	Specifies the size of the *data* buffer.
buf	Specifies the buffer to store data.
buf_len	Specifies the length of *buffer*.
predicate	Specifies the predicate for the XIM data.
predicate_arg	Specifies the predicate specific data.

The predicate procedure indicates whether the *data* is for the XIM or not. *len* This function stores the read data in *buf*, which size is specified as *buf_len*. The size of data is set to *ret_len*. If *preedicate()* returns True, this function returns True. If not, it calls the registered callback function.

The procedure and its arguments are:

```
Bool (*predicate)(im, len, data, predicate_arg)
    XIM im;
    INT16 len;
    XPointer data;
    XPointer predicate_arg;
```

im	Specifies XIM structure address.
len	Specifies the size of the *data* buffer.
data	Specifies the buffer to store data.

predicate_arg Specifies the predicate specific data.

3.5. Flushing buffer

The following function is called when Xlib needs to flush the data.

void (*flush)(*im*)
    XIM *im*;

im Specifies XIM structure address.

This function must flush the data stored in internal buffer on the transport layer. If data transfer is completed, the function returns True. The Alternative Entry for this function is:

void _XimFlush(*im*)
    XIM *im*;

im Specifies XIM structure address.

3.6. Registering asynchronous data handler

Xlib needs to handle asynchronous response from IM Server. This is because some of the XIM data occur asynchronously to X events.

Those data will be handled in the *Filter*, and the *Filter* will call asynchronous data handler in the protocol layer. Then it calls dispatchers in the transport layer. The dispatchers are implemented by the protocol layer. This function must store the information and prepare for later call of the dispatchers using **_XimCallDispatcher**.

When multiple dispatchers are registered, they will be called sequentially in order of registration, on arrival of asynchronous data. The register_dispatcher is declared as following:

Bool (*register_dispatcher)(*im*, *dispatcher*, *call_data*)
    XIM *im*;
    Bool *(*dispatcher)()*;
    XPointer *call_data*;

im Specifies XIM structure address.

dispatcher Specifies the dispatcher function to register.

call_data Specifies a parameter for the *dispatcher*.

The dispatcher is a function of the following type:

Bool (*dispatcher)(*im*, *len*, *data*, *call_data*)
    XIM *im*;
    INT16 *len*;
    XPointer *data*;
    XPointer *call_data*;

im Specifies XIM structure address.

len Specifies the size of the *data* buffer.

data Specifies the buffer to store data.

call_data Specifies a parameter passed to the register_dispatcher.

The dispatcher is provided by the protocol layer. They are called once for every asynchronous data, in order of registration. If the data is used, it must return True. otherwise, it must return False.

If the dispatcher function returns True, the Transport Layer assume that the data has been processed by the upper layer. The Alternative Entry for this function is:

Bool _XimRegisterDispatcher(*im*, *dispatcher*, *call_data*)
    XIM *im*;
    Bool *(*dispatcher)()*;
    XPointer *call_data*;

im	Specifies XIM structure address.
dispatcher	Specifies the dispatcher function to register.
call_data	Specifies a parameter for the *dispatcher*.

3.7. Calling dispatcher

The following function is used to call the registered dispatcher function, when the asynchronous response from IM Server has arrived.

Bool (*call_dispatcher)(*im*, *len*, *data*)
    XIM *im*;
    INT16 *len*;
    XPointer *data*;

im	Specifies XIM structure address.
len	Specifies the size of *data* buffer.
data	Specifies the buffer to store data.

The call_dispatcher must call the dispatcher function, in order of their registration. *len* and *data* are the data passed to register_dispatcher.

The return values are checked at each invocation, and if it finds True, it immediately return with true for its return value.

It is depend on the upper layer whether the read data is XIM Protocol packet unit or not. The Alternative Entry for this function is:

Bool _XimCallDispatcher(*im*, *len*, *data*)
    XIM *im*;
    INT16 *len*;
    XPointer *call_data*;

4. Sample implementations for the Transport Layer

Sample implementations for the transporter using the X connection is described here.

4.1. X Transport

At the beginning of the X Transport connection for the XIM transport mechanism, two different windows must be created either in an Xlib XIM or in an IM Server, with which the Xlib and the IM Server exchange the XIM transports by using the ClientMessage events and Window Properties. In the following, the window created by the Xlib is referred as the "client communication window", and on the other hand, the window created by the IM Server is referred as the "IMS communication window".

4.1.1. Connection

In order to establish a connection, a communication window is created. A ClientMessage in the following event's format is sent to the owner window of XIM_SERVER selection, which the IM Server has created.

Refer to "The Input Method Protocol" for the XIM_SERVER atom.

Table 4-1; The ClientMessage sent to the IMS window.

Structure Member		Contents
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	IMS Window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_XCONNECT’’, False)
int	format	32
long	data.l[0]	client communication window ID
long	data.l[1]	client-major-transport-version (*1)
long	data.l[2]	client-major-transport-version (*1)

In order to establish the connection (to notify the IM Server communication window), the IM Server sends a ClientMessage in the following event's format to the client communication window.

Table 4-2; The ClientMessage sent by IM Server.

Structure Member		Contents
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	client communication window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_XCONNECT’’, False)
int	format	32
long	data.l[0]	IMS communication window ID
long	data.l[1]	server-major-transport-version (*1)
long	data.l[2]	server-minor-transport-version (*1)
long	data.l[3]	dividing size between ClientMessage and Property (*2)

(*1) major/minor-transport-version

The read/write method is decided by the combination of major/minor-transport-version, as follows:

Table 4-3; The read/write method and the major/minor-transport-version

Transport-version		**read/write**
major	**minor**	
0	0 1 2	only-CM & Property-with-CM only-CM & multi-CM only-CM & multi-CM & Property-with-CM
1	0	PropertyNotify
2	0 1	only-CM & PropertyNotify only-CM & multi-CM & PropertyNotify

only-CM	:	data is sent via a ClientMessage
multi-CM	:	data is sent via multiple ClientMessages
Property-with-CM	:	data is written in Property, and its Atom is send via ClientMessage
PropertyNotify	:	data is written in Property, and its Atom is send via PropertyNotify

The method to decide major/minor-transport-version is as follows:

(1) The client sends 0 as major/minor-transport-version to the IM Server. The client must support all methods in Table 4-3. The client may send another number as major/minor-transport-version to use other method than the above in the future.

(2) The IM Server sends its major/minor-transport-version number to the client. The client sends data using the method specified by the IM Server.

(3) If major/minor-transport-version number is not available, it is regarded as 0.

(*2) dividing size between ClientMessage and Property

If data is sent via both of multi-CM and Property, specify the dividing size between ClientMessage and Property. The data, which is smaller than this size, is sent via multi-CM (or only-CM), and the data, which is lager than this size, is sent via Property.

4.1.2. read/write

The data is transferred via either ClientMessage or Window Property in the X Window System.

4.1.2.1. Format for the data from the Client to the IM Server

ClientMessage

If data is sent via ClientMessage event, the format is as follows:

Table 4-4; The ClientMessage event's format (first or middle)

Structure Member	**Contents**

Structure Member		Contents
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	IMS communication window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_MOREDATA’’, False)
int	format	8
char	data.b[20]	(read/write DATA : 20 byte)

Table 4-5; The ClientMessage event’s format (only or last)

Structure Member		Contents
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	IMS communication window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_PROTOCOL’’, False)
int	format	8
char	data.b[20]	(read/write DATA : MAX 20 byte) (*1)

(*1) If the data is smaller than 20 byte, all data other than available data must be 0.

Property

In the case of large data, data will be sent via the Window Property for the efficiency. There are the following two methods to notify Property, and transport-version is decided which method is used.

(1) The XChangeProperty function is used to store data in the client communication window, and Atom of the stored data is notified to the IM Server via ClientMessage event.

(2) The XChangeProperty function is used to store data in the client communication window, and Atom of the stored data is notified to the IM Server via PropertyNotify event.

The arguments of the XChangeProperty are as follows:

Table 4-6; The XChangeProperty event’s format

Argument		Contents
Display	*display	The display to which connects
Window	window	IMS communication window ID
Atom	property	read/write property Atom (*1)
Atom	type	XA_STRING
int	format	8
int	mode	PropModeAppend
u_char	*data	read/write DATA
int	nelements	length of DATA

(*1) The read/write property ATOM allocates the following strings by **XInternAtom**.
‘‘_clientXXX’’

The client changes the property with the mode of PropModeAppend and the IM Server will read it with the delete mode i.e. (delete = True).

If Atom is notified via ClientMessage event, the format of the ClientMessage is as follows:

Table 4-7; The ClientMessage event's format to send Atom of property

Structure Member		**Contents**
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	IMS communication window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_PROTOCOL’’, False)
int	format	32
long	data.l[0]	length of read/write property Atom
long	data.l[1]	read/write property Atom

4.1.2.2. Format for the data from the IM Server to the Client

ClientMessage

The format of the ClientMessage is as follows:

Table 4-8; The ClientMessage event's format (first or middle)

Structure Member		**Contents**
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	client communication window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_MOREDATA’’, False)
int	format	8
char	data.b[20]	(read/write DATA : 20 byte)

Table 4-9; The ClientMessage event's format (only or last)

Structure Member		**Contents**
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	client communication window ID
Atom	message_type	XInternAtom(display, ‘‘_XIM_PROTOCOL’’, False)
int	format	8
char	data.b[20]	(read/write DATA : MAX 20 byte) (*1)

(*1) If the data size is smaller than 20 bytes, all data other than available data must be 0.

Property

In the case of large data, data will be sent via the Window Property for the efficiency. There are the following two methods to notify Property, and transport-version is decided which method is used.

(1) The XChangeProperty function is used to store data in the IMS communication window, and Atom of the property is sent via the ClientMessage event.

(2) The XChangeProperty function is used to store data in the IMS communication window, and Atom of the property is sent via PropertyNotify event.

The arguments of the XChangeProperty are as follows:

Table 4-10; The XChangeProperty event's format

Argument		Contents
Display	*display	The display which to connects
Window	window	client communication window ID
Atom	property	read/write property Atom (*1)
Atom	type	XA_STRING
int	format	8
int	mode	PropModeAppend
u_char	*data	read/write DATA
int	nelements	length of DATA

(*1) The read/write property ATOM allocates some strings, which are not allocated by the client, by **XInternAtom**.

The IM Server changes the property with the mode of PropModeAppend and the client reads it with the delete mode, i.e. (delete = True).

If Atom is notified via ClientMessage event, the format of the ClientMessage is as follows:

Table 4-11; The ClientMessage event's format to send Atom of property

Structure Member		Contents
int	type	ClientMessage
u_long	serial	Set by the X Window System
Bool	send_event	Set by the X Window System
Display	*display	The display to which connects
Window	window	client communication window ID
Atom	message_type	XInternAtom(display, "_XIM_PROTOCOL", False)
int	format	32
long	data.l[0]	length of read/write property ATOM
long	data.l[1]	read/write property ATOM

4.1.3. Closing Connection

If the client disconnect with the IM Server, shutdown function should free the communication window properties and etc..

5. References

[1] Masahiko Narita and Hideki Hiura, *"The Input Method Protocol"*