The Input Method Protocol

X Consortium Standard

Masahiko Narita, FUJITSU Limited.
Hideki Hiura, SunSoft, Inc.

X Version 11, Release 7.7

Version 1.0

Copyright © 1993, 1994 FUJITSU LIMITED, Oracle and/or its affiliates

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

Copyright © 1993, 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 The Open Group.

Abstract

This specifies a protocol between IM library and IM (Input Method) Server for internationalized text input, which is indepedent from any specific language, any specific input method and the transport layer used in communication between the IM library and the IM Server, and uses a client-server model. This protocol allows user to use his/her favorite method for all applications within the stand-along distrubuted environment.

Table of Contents

Introduction

Scope

The internationalization in the X Window System Version 11, Release 5 (X11R5) provides a common API which application developers can use to create portable internationalized programs and to adapt them to the requirements of different native languages, local customs, and character string encodings (this is called "localization"). As one of its internationalization mechanisms X11R5 has defined a functional interface for internationalized text input, called XIM (X Input Method).

When a client-server model is used with an IM (Input Method) implementation, a protocol must be established between the client and the server. However, the protocol used to interface Input Method Servers (IM Servers) with the Input Method libraries (IM libraries) to which applications are linked was not addressed in X11R5. This led application developers to depend on vendor-specific input methods, decreased the user's choice of available input methods, and made it more difficult for developers to create portable applications. This paper describes the Input Method Protocol developed for X11R6 to resolve the above problems and to address the requirements of existing and future input methods.

The Input Method Protocol is independent from the transport layer used in communication between the IM library and the IM Server. Thus, the input method protocol can be built on any inter-process communication mechanism, such as TCP/IP or the X protocol.

In addition, the protocol provides for future extensions such as differing input model types.

Background

Text input is much more simple for some languages than others. English, for instance, uses an alphabet of a manageable size, and input consists of pressing the corresponding key on a keyboard, perhaps in combination with a shift key for capital letters or special characters.

Some languages have larger alphabets, or modifiers such as accents, which require the addition of special key combinations in order to enter text. These input methods may require "dead-keys" or "compose-keys" which, when followed by different combinations of key strokes, generate different characters.

Text input for ideographic languages is much less simple. In these languages, characters represent actual objects rather than phonetic sounds used in pronouncing a word, and the number of characters in these languages may continue to grow. In Japanese, for instance, most text input methods involve entering characters in a phonetic alphabet, after which the input method searches a dictionary for possible ideographic equivalents (of which there may be many). The input method then presents the candidate characters for the user to choose from.

In Japanese, either Kana (phonetic symbols) or Roman letters are typed and then a region is selected for conversion to Kanji. Several Kanji characters may have the same phonetic representation. If that is the case with the string entered, a menu of characters is presented and the user must choose the appropriate one. If no choice is necessary or a preference has been established, the input method does the substitution directly.

These complicated input methods must present state information (Status Area), text entry and edit space (Preedit Area), and menu/choice presentations (Auxiliary Area). Much of the protocol between the IM library and the IM Server involves managing these IM areas. Because of the size and complexity of these input methods, and because of how widely they vary from one language or locale to another, they are usually implemented as separate processes which can serve many client processes on the same computer or network.

Input Method Styles

X11 internationalization support includes the following four types of input method:

- on-the-spot:	The client application is directed by the IM Server to display all pre-edit data at the site of text insertion. The client registers callbacks invoked by the input method during pre-editing.
- off-the-spot:	The client application provides display windows for the pre-edit data to the input method which displays into them directly.
- over-the-spot:	The input method displays pre-edit data in a window which it brings up directly over the text insertion position.
- root-window:	The input method displays all pre-edit data in a separate area of the screen in a window specific to the input method.

Client applications must choose from the available input methods supported by the IM Server and provide the display areas and callbacks required by the input method.

Architecture

Implementation Model

Within the X Window System environment, the following two typical architectural models can be used as an input method's implementation model.

- Client/Server model:	A separate process, the IM Server, processes input and handles preediting, converting, and committing. The IM library within the application, acting as client to the IM Server, simply receives the committed string from the IM Server.
- Library model:	All input is handled by the IM library within the application. The event process is closed within the IM library and a separate IM Server process may not be required.

Most languages which need complex preediting, such as Asian languages, are implemented using the Client/Server IM model. Other languages which need only dead key or compose key processing, such as European languages, are implemented using the Library model.

In this paper, we discuss mainly the Client/Server IM model and the protocol used in communication between the IM library (client) and the IM Server.

Structure of IM

When the client connects or disconnects to the IM Server, an open or close operation occurs between the client and the IM Server.

The IM can be specified at the time of XOpenIM() by setting the locale of the client and a locale modifier. Since the IM remembers the locale at the time of creation XOpenIM() can be called multiple times (with the setting for the locale and the locale modifier changed) to support multiple languages.

In addition, the supported IM type can be obtained using XGetIMValues().

The client usually holds multiple input (text) fields. Xlib provides a value type called the "Input Context" (IC) to manage each individual input field. An IC can be created by specifying XIM using XCreateIC(), and it can be destroyed using XDestroyIC().

The IC can specify the type of IM which is supported by XIM for each input field, so each input field can handle a different type of IM.

Most importantly information such as the committed string sent from the IM Server to the client, is exchanged based on each IC.

Since each IC corresponds to an input field, the focused input field should be announced to the IM Server using XSetICFocus(). (XUnsetICFocus() can also be used to change the focus.)

Event Handling Model

Existing input methods support either the FrontEnd method, the BackEnd method, or both. This protocol specifically supports the BackEnd method as the default method, but also supports the FrontEnd method as an optional IM Server extension.

The difference between the FrontEnd and BackEnd methods is in how events are delivered to the IM Server. (Fig. 1)

BackEnd Method

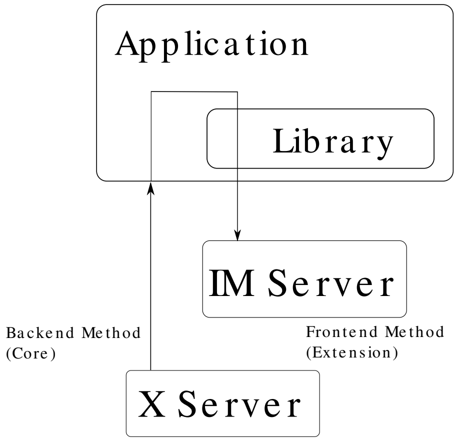
In the BackEnd method, client window input events are always delivered to the IM library, which then passes them to the IM Server. Events are handled serially in the order delivered, and therefore there is no synchronization problem between the IM library and the IM Server.

Using this method, the IM library forwards all KeyPress and KeyRelease events to the IM Server (as required by the Event Flow Control model described in Event Flow Control) and synchronizes with the IM Server (as described in Filtering Events).

FrontEnd Method

In the FrontEnd method, client window input events are delivered by the X server directly to both the IM Server and the IM library. Therefore this method provides much better interactive performance while preediting (particularly in cases such as when the IM Server is running locally on the user's workstation and the client application is running on another workstation over a relatively slow network).

However, the FrontEnd model may have synchronization problems between the key events handled in the IM Server and other events handled in the client, and these problems could possibly cause the loss or duplication of key events. For this reason, the BackEnd method is the core method supported, and the FrontEnd method is made available as an extension for performance purposes. (Refer to *Common Extensions* for more information.)



The flow of events

Event Flow Control

This protocol supports two event flow models for communication between the IM library and the IM Server (Static and Dynamic).

Static Event Flow requires that input events always be sent to the IM Server from the client.

Dynamic Event Flow, however, requires only that those input events which need to be processed (converted) be sent to the IM Server from the client.

For instance, in the case of inputing a combination of ASCII characters and Chinese characters, ASCII characters do not need to be processed in the IM Server, so their key events do not have to be sent to the IM Server. On the other hand, key events necessary for composing Chinese characters must be sent to the IM Server.

Thus, by adopting the Dynamic Event Flow, the number of requests among the X Server, the client, and the IM Server is significantly reduced, and the number of context switches is also reduced, resulting in improved performance. The IM Server can send `XIM_REGISTER_TRIGGERKEYS` message in order to switch the event flow in the Dynamic Event Flow.

The protocol for this process is described in Event Flow Control.

Default Preconnection Convention

IM Servers are strongly encouraged to register their symbolic names as the ATOM names into the IM Server directory property, `XIM_SERVERS`, on the root window of the screen_number 0. This property can contain a list of ATOMs, and the each ATOM represents each possible IM Server. IM Server names are restricted to POSIX Portable Filename Character Set. To discover if the IM Server is active, see if there is an owner for the selection with that atom name. To learn the address of that IM Server, convert the selection target `TRANSPORT`, which will return a string form of the transport address(es). To learn the supported locales of that IM Server, convert the selection target `LOCALES`, which will return a set of names of the supported locales in the syntax X/Open defines.

The basic semantics to determine the IM Server if there are multiple ATOMs are found in `XIM_SERVERS` property, is first fit if the IM Server name is not given as a X modifier's category `im`.

The address information retrievable from the `TRANSPORT` target is a transport-specific name. The preregistered formats for transport-specific names are listed in *Transport List*. Additional transport-specific names may be registered with X Consortium.

For environments that lack X connections, or for IM Servers which do not use the X Window System, the preconnection convention with IM Server may be given outside the X Window system (e.g. using a Name Service).

Protocol

The protocol described below uses the bi-directional synchronous/asynchronous request/reply/error model and is specified using the same conventions outlined in Section 2 of the core X Window System protocol [1]:

Basic Requests Packet Format

This section describes the requests that may be exchanged between the client and the IM Server.

The basic request packet header format is as follows.

```
major-opcode:                  CARD8
minor-opcode:                  CARD8
length:                        CARD16
```

The MAJOR-OPCODE specifies which core request or extension package this packet represents. If the MAJOR-OPCODE corresponds to a core request, the MINOR-OPCODE contains 8 bits of request-specific data. (If the MINOR-OPCODE is not used, it is 0.) Otherwise, the MAJOR-OPCODE and the MINOR-OPCODE are specified by `XIM_QUERY_EXTENSION` message. (Refer to 4.7. Query the supported extension protocol list.) The LENGTH field specifies the number of 4 bytes elements following the header. If no additional data is followed by the header, the LENGTH field will be 0.

Data Types

The following data types are used in the core X IM Server protocol:

```
BITMASK16
     CARD16
BITMASK32
     CARD32
PADDING FORMAT
     Where N is some expression, and Pad(N) is the number of bytes needed to round

          Pad(N) = (4 - (N mod 4)) mod 4

LPCE
  1         A character from the4 X Portable Character Set in Latin Portable
            Character Encoding

STRING
     2     n              length of string in bytes
     n     LISTofLPCE     string
     p                    unused, p=Pad(2+n)

STR
     1     n              length of name in bytes
     n     STRING8        name

XIMATTR
     2     CARD16         attribute ID (*1)
     2     CARD16         type of the value (*2)
     2     n              length of im-attribute
     n     STRING8        im-attribute
     p                    unused, p = Pad(2+n)
```

The im-attribute argument specifies XIM values such as XNQueryInputStyle.

```
XICATTR
     2     CARD16         attribute ID (*1)
     2     CARD16         type of the value (*2)
     2     n              length of ic-attribute
     n     STRING8        ic-attribute
     p                    unused, p = Pad(2+n)
```

(*1) XIMATTR and XICATTR are used during the setup stage and XIMATTRIBUTE and XICATTRIBUTE are used after each attribute ID has been recognized by the IM Server and the IM library.

(*2) The value types are defined as follows:

values	data	format
#0	Separator of NestedList	-----(*3)
#1	byte data	CARD8
#2	word data	CARD16
#3	long data	CARD32
#4	char data	STRING8

values	**data**	**format**		
#5	Window	CARD32		
#10	XIMStyles	2	n	number of XIMStyle list
		2		unused
		n	CARD32	XIMStyle list
#11	XRectangle	2	INT16	X
		2	INT16	Y
		2	CARD16	width
		2	CARD16	height
#12	XPoint	2	INT16	X
		2	INT16	Y
#13	XFontSet	2	n	length of Base font name
		n	STRING8	Base font name list
		p		unused, p = Pad(2+n)
#15	XIMHotKeyTriggers	4	n	number of XIMTRIGGERKEY list (*4)
		n	XIMTRIGGERKEY	XIMHotkeyTrigger list
		n	XIMHOTKEYSTATE	HotKey processing state
#17	XIMStringConversion	XIMSTRCONVTEXT		
#18	XIMPreeditState	XIMPREEDITSTATE		
#19	XIMResetState	XIMRESETSTATE		
#x7fff	NestedList	-----		

```
(*3)  The IC value for the separator of NestedList is defined as follows,
            #define   XNSeparatorofNestedList   "separatorofNestedList"
      , which is registered in X Consortium and cannot be used for any other purpo

(*4) LISTofFOO
            A Type name of the form LISTof FOO means a counted list of elements of
            The size of the length field may vary (it is not necessarily the same
            size as a FOO), and in some cases, it may be implicit.
XIMTRIGGERKEY
     4     CARD32       keysym
     4     CARD32       modifier
     4     CARD32       modifier mask
```

```
ENCODINGINFO
     2     n            length of encoding info
     n     STRING8      encoding info
     p                  unused, p=Pad(2+n)


     1     CARD8        extension major-opcode
     1     CARD8        extension minor-opcode
     2     n            length of extension name
     n     STRING8      extension name
     p                  unused, p = Pad(n)

XIMATTRIBUTE
     2     CARD16       attribute ID
     2     n            value length
     n                  value
     p                  unused, p = Pad(n)

XICATTRIBUTE
     2     CARD16       attribute ID
     2     n            value length
     n                  value
     p                  unused, p = Pad(n)



XIMSTRCONVTEXT
     2    CARD16                       XIMStringConversionFeedback
          #x0000001                    XIMStringConversionLeftEdge
          #x0000002                    XIMStringConversionRightEdge
          #x0000004                    XIMStringConversionTopEdge
          #x0000008                    XIMStringConversionBottomEdge
          #x0000010                    XIMStringConversionConvealed
          #x0000020                    XIMStringConversionWrapped
     2     n                           byte length of the retrieved string
     n     STRING8                     retrieved string
     p                                 unused, p = Pad(n)
     2     m                           byte length of feedback array
     2                                 unused
     m     LISTofXIMSTRCONVFEEDBACK   feedback array(*1)

(*1)   This field is reserved for future use.

XIMFEEDBACK
     4    CARD32       XIMFeedback
          #x000001     XIMReverse
          #x000002     XIMUnderline
          #x000004     XIMHighlight
          #x000008     XIMPrimary
          #x000010     XIMSecondary
          #x000020     XIMTertiary
          #x000040     XIMVisibleToForward
          #x000080     XIMVisibleToBackward
          #x000100     XIMVisibleCenter
```

```
XIMHOTKEYSTATE
     4    CARD32       XIMHotKeyState
          #x0000001    XIMHotKeyStateON
          #x0000002    XIMHotKeyStateOFF

XIMPREEDITSTATE
     4    CARD32       XIMPreeditState
          #x0000001    XIMPreeditEnable
          #x0000002    XIMPreeditDisable

XIMRESETSTATE
     4    CARD32       XIMResetState
          #x0000001    XIMInitialState
          #x0000002    XIMPreserveState
```

Error Notification

Both the IM Server and the IM library return `XIM_ERROR` messages instead of the corresponding reply messages if any errors occur during data processing.

At most one error is generated per request. If more than one error condition is encountered in processing a request, the choice of which error is returned is implementation-dependent.

```
XIM_ERROR (IM Server <--> IM library)

          2     CARD16          input-method-ID
          2     CARD16          input-context-ID
          2     BITMASK16       flag (*1)
                #0000           Both Input-Method-ID and Input-Context-ID are inva
                #0001           Input-Method-ID is valid
                #0002           Input-Context-ID is valid
          2     CARD16          Error Code
                #1              BadAlloc
                #2              BadStyle
                #3              BadClientWindow
                #4              BadFocusWindow
                #5              BadArea
                #6              BadSpotLocation
                #7              BadColormap
                #8              BadAtom
                #9              BadPixel
                #10             BadPixmap
                #11             BadName
                #12             BadCursor
                #13             BadProtocol
                #14             BadForeground
                #15             BadBackground
                #16             LocaleNotSupported
                #999            BadSomething (*2)
          2     n               byte length of error detail.
          2     CARD16          type of error detail (*3)
```

```
          n     STRING8         error detail (*4)
          p                     unused, p = Pad(n)

(*1)  Before an IM is created, both Input-Method-ID and
      Input-Context-ID are invalid.
      Before an IC is created, only Input-Method-ID is valid.
      After that, both of Input-Method-ID and Input-Context-ID are valid.

(*2) Unspecific error, for example "language engine died"

(*3) This field is reserved for future use.

(*4) Vendor defined detail error message
```

Connection Establishment

`XIM_CONNECT` message requests to establish a connection over a mutually-understood virtual stream.

```
XIM_CONNECT (IM library -> IM Server)
     1                      byte order
           #x42             MSB first
           #x6c             LSB first
     1                      unused
     2     CARD16           client-major-protocol-version (*1)
     2     CARD16           client-minor-protocol-version (*1)
     2     CARD16           number of client-auth-protocol-names
     n     LISTofSTRING     client-auth-protocol-names

(*1) Specify the version of IM Protocol that the client supports.
```

A client must send `XIM_CONNECT` message as the first message on the connection. The list specifies the names of authentication protocols the sending IM Server is willing to perform. (If the client need not authenticate, the list may be omited.)

`XIM_AUTH_REQUIRED` message is used to send the authentication protocol name and protocol-specific data.

```
XIM_AUTH_REQUIRED (IM library <--> IM Server)

     1     CARD8              auth-protocol-index
     3                        unused
     2     n                  length of authentication data
     2                        unused
     n     <varies>     data
     p                        unused, p = Pad(n)
```

The auth-protocol is specified by an index into the list of names given in the `XIM_CONNECT` or `XIM_AUTH_SETUP` message. Any protocol-specific data that might be required is also sent.

The IM library sends `XIM_AUTH_REPLY` message as the reply to `XIM_AUTH_REQUIRED` message, if the IM Server is authenticated.

```
XIM_AUTH_REPLY (IM library -> IM Server)
     2     n                 length of authentication data
     2                       unused
     2     n                 length of authentication data
     2                       unused
     n     <varies>      data
     p                       unused, p = Pad(n)
```

The auth data is specific to the authentication protocol in use.

`XIM_AUTH_NEXT` message requests to send more auth data.

```
XIM_AUTH_NEXT (IM library <--> IM Server)
     2     n                 length of authentication data
     2                       unused
     n     <varies>      data
     p                       unused, p = Pad(n)
```

The auth data is specific to the authentication protocol in use.

The IM Server sends `XIM_AUTH_SETUP` message to authenticate the client.

```
XIM_AUTH_SETUP (IM Server -> IM library)
     2     CARD16           number of client-auth-protocol-names
     2                      unused
     n     LISTofSTRING     server-auth-protocol-names
```

The list specifies the names of authentication protocols the client is willing to perform.

`XIM_AUTH_NG` message requests to give up the connection.

XIM_AUTH_NG (IM library <--> IM Server)

The IM Server sends `XIM_CONNECT_REPLY` message as the reply to `XIM_CONNECT` or `XIM_AUTH_REQUIRED` message.

```
XIM_CONNECT_REPLY (IM Server -> IM library)
     2     CARD16     server-major-protocol-version (*1)
     2     CARD16     server-minor-protocol-version (*1)
```

```
(*1) Specify the version of IM Protocol that the IM Server supports.
This document specifies major version one, minor version zero.
```

Here are the state diagrams for the client and the IM Server.

State transitions for the client

init_status: Use authorization function -> *client_ask*

Not use authorization function -> *client_no_check*

start: Send `XIM_CONNECT`

If *client_ask* -> *client_wait1*

If *client_no_check*, client-auth-protocol-names may be omited -> *client_wait2*

client_wait1: Receive `XIM_AUTH_REQUIRED` -> *client_check*

Receive <other> -> *client_NG*

client_check: If no more auth needed, send `XIM_AUTH_REPLY` -> *client_wait2*

If good auth data, send `XIM_AUTH_NEXT` -> *client_wait1*

If bad auth data, send `XIM_AUTH_NG` -> give up on this protocol

client_wait2: Receive `XIM_CONNECT_REPLY` -> connect Receive `XIM_AUTH_SETUP` -> *client_more*

Receive `XIM_AUTH_NEXT` -> *client_more*

Receive `XIM_AUTH_NG` -> give up on this protocol

Receive <other> -> *client_NG*

client_more: Send `XIM_AUTH_REQUIRED` -> *client_wait2*

client_NG: Send `XIM_AUTH_NG` -> give up on this protocol

State transitions for the IM Server

init_status: Use authorization function -> *server_ask*

Not use authorization function -> *server_no_check*

start: Receive `XIM_CONNECT`

-> *start2* Receive <other> -> *server_NG*

start2: If *client_ask*, send `XIM_AUTH_REQUIRED` -> *server_wait1*

If *client_no_check* and *server_ask*, send `XIM_AUTH_SETUP` -> *server_wait2*

If *client_no_check* and *server_no_check*, send `XIM_CONNECT_REPLY` -> connect

server_wait1: Receive `XIM_AUTH_REPLY` -> *server2*

Receive `XIM_AUTH_NEXT` -> *server_more*

Receive <other> -> *server_NG*

server_more Send `XIM_AUTH_REQUIRED` -> *server_wait1*

server2 If *server_ask*, send `XIM_AUTH_SETUP` -> *server_wait2*

If *server_no_check*, send `XIM_CONNECT_REPLY` -> connect

server_wait2	Receive `XIM_AUTH_REQUIRED` -> *server_check*
	Receive <other> -> *server_NG*
server_check	If no more auth data, send `XIM_CONNECT_REPLY` -> connect
	If bad auth data, send `XIM_AUTH_NG` -> give up on this protocol
	If good auth data, send `XIM_AUTH_NEXT` -> *server_wait2*
server_NG	Send `XIM_AUTH_NG` -> give up on this protocol

`XIM_DISCONNECT` message requests to shutdown the connection over a mutually-understood virtual stream.

XIM_DISCONNECT (IM library -> IM Server)

`XIM_DISCONNECT` is a synchronous request. The IM library should wait until it receives either an `XIM_DISCONNECT_REPLY` packet or an `XIM_ERROR` packet.

XIM_DISCONNECT_REPLY (IM Server -> IM library)

`XIM_OPEN` requests to establish a logical connection between the IM library and the IM Server.

```
XIM_OPEN (IM library -> IM Server)
     n     STR     locale name
     p             unused, p = Pad(n)
```

`XIM_OPEN` is a synchronous request. The IM library should wait until receiving either an `XIM_OPEN_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_OPEN_REPLY (IM Server -> IM library)
     2     CARD16             input-method-ID
     2     n                  byte length of IM attributes supported
     n     LISTofXIMATTR      IM attributes supported
     2     m                  byte length of IC attributes supported
     2     CARD16             unused
     m     LISTofXICATTR      IC attributes supported
```

`XIM_OPEN_REPLY` message returns all supported IM and IC attributes in LISTofXIMATTR and LISTofXICATTR. These IM and IC attribute IDs are used to reduce the amount of data which must be transferred via the network. In addition, this indicates to the IM library what kinds of IM/IC attributes can be used in this session, and what types of data will be exchanged. This allows the IM Server provider and application writer to support IM system enhancements with new IM/IC attributes, without modifying Xlib. The IC value for the separator of NestedList must be included in the LISTofXICATTR.

`XIM_CLOSE` message requests to shutdown the logical connection between the IM library and the IM Server.

```
XIM_CLOSE (IM library -> IM Server)
```

```
     2     CARD16     input-method-ID
     2                unused
```

`XIM_CLOSE` is a synchronous request. The IM library should wait until receiving either an `XIM_CLOSE_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_CLOSE_REPLY (IM Server -> IM library)
     2     CARD16     input-method-ID
     2                unused
```

Event Flow Control

An IM Server must send `XIM_SET_EVENT_MASK` message to the IM library in order for events to be forwarded to the IM Server, since the IM library initially doesn't forward any events to the IM Server. In the protocol, the IM Server will specify masks of X events to be forwarded and which need to be synchronized by the IM library.

```
XIM_SET_EVENT_MASK (IM Server -> IM library)
     2     CARD16        input-method-ID
     2     CARD16        input-context-ID
     4     EVENTMASK     forward-event-mask (*1)
     4     EVENTMASK     synchronous-event-mask (*2)

(*1)  Specify all the events to be forwarded to the IM Server by the IM library.
(*2)  Specify the events to be forwarded with synchronous flag on by the IM librar
```

`XIM_SET_EVENT_MASK` is an asynchronous request. The event masks are valid immediately after they are set until changed by another `XIM_SET_EVENT_MASK` message. If input-context-ID is set to zero, the default value of the input-method-ID will be changed to the event masks specified in the request. That value will be used for the IC's which have no individual values.

Using the Dynamic Event Flow model, an IM Server sends `XIM_REGISTER_TRIGGERKEYS` message to the IM library before sending `XIM_OPEN_REPLY` message. Or the IM library may suppose that the IM Server uses the Static Event Flow model.

```
XIM_REGISTER_TRIGGERKEYS (IM Server -> IM library)

     2     CARD16                  input-method-ID
     2                             unused
     4     n                       byte length of on-keys
     n     LISTofXIMTRIGGERKEY     on-keys list
     4     m                       byte length of off-keys
     m     LISTofXIMTRIGGERKEY     off-keys list
```

`XIM_REGISTER_TRIGGERKEYS` is an asynchronous request. The IM Server notifys the IM library of on-keys and off-keys lists with this message.

The IM library notifys the IM Server with `XIM_TRIGGER_NOTIFY` message that a key event matching either on-keys or off-keys has been occurred.

```
XIM_TRIGGER_NOTIFY (IM library -> IM Server)
     2     CARD16           input-method-ID
     2     CARD16           input-context-ID
     4     CARD32           flag
           #0               on-keys list
           #1               off-keys list
     4     CARD32           index of keys list
     4     EVENTMASK        client-select-event-mask (*1)

(*1)  Specify the events currently selected by the IM library with XSelectInput.
```

`XIM_TRIGGER_NOTIFY` is a synchronous request. The IM library should wait until receiving either an `XIM_TRIGGER_NOTIFY_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_TRIGGER_NOTIFY_REPLY (IM Server -> IM library)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

Encoding Negotiation

`XIM_ENCODING_NEGOTIATION` message requests to decide which encoding to be sent across the wire. When the negotiation fails, the fallback default encoding is Portable Character Encoding.

```
XIM_ENCODING_NEGOTIATION (IM library -> IM Server).sp 6p
     2     CARD16                 input-method-ID
     2     n                      byte length of encodings listed by name
     n     LISTofSTR              list of encodings supported in the IM library.
     p                            unused, p = Pad(n)
     2     m                      byte length of encodings listed by detailed data
     2                            unused
     m     LISTofENCODINGINFO     list of encordings supported in the IM library
```

The IM Server must choose one encoding from the list sent by the IM library. If index of the encording determined is -1 to indicate that the negotiation is failed, the fallback default encoding is used. The message must be issued after sending `XIM_OPEN` message via XOpenIM(). The name of encoding may be registered with X Consortium.

`XIM_ENCODING_NEGOTIATION` is a synchronous request. The IM library should wait until receiving either an `XIM_ENCODING_NEGOTIATION_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_ENCODING_NEGOTIATION_REPLY (IM Server -> IM library)
     2     CARD16     input-method-ID
     2     CARD16     category of the encoding determined.
           #0         name
           #1         detailed data
     2     INT16      index of the encoding determinated.
```

```
     2                 unused
```

Query the supported extension protocol list

`XIM_QUERY_EXTENSION` message requests to query the IM extensions supported by the IM Server to which the client is being connected.

```
XIM_QUERY_EXTENSION (IM library -> IM Server)
     2     CARD16        input-method-ID
     2     n             byte length of extensions supported by the IM library
     n     LISTofSTR     extensions supported by the IM library
     p                   unused, p = Pad(n)
```

An example of a supported extension is FrontEnd. The message must be issued after sending `XIM_OPEN` message via XOpenIM().

If n is 0, the IM library queries the IM Server for all extensions.

If n is not 0, the IM library queries whether the IM Server supports the contents specified in the list.

If a client uses an extension request without previously having issued a `XIM_QUERY_EXTENSION` message for that extension, the IM Server responds with a `BadProtocol` error. If the IM Server encounters a request with an unknown MAJOR-OPCODE or MINOR-OPCODE, it responds with a `BadProtocol` error.

`XIM_QUERY_EXTENSION` is a synchronous request. The IM library should wait until receiving either an `XIM_QUERY_EXTENSION_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_QUERY_EXTENSION_REPLY (IM Server -> IM library)
     2     CARD16      input-method-ID
     2     n           byte length of extensions supported by both the IM library
     n     LISTofEXT   list of extensions supported by both the IM library and the
```

`XIM_QUERY_EXTENSION_REPLY` message returns the list of extensions supported by both the IM library and the IM Server. If the list passed in `XIM_QUERY_EXTENSION` message is NULL, the IM Server returns the full list of extensions supported by the IM Server. If the list is not NULL, the IM Server returns the extensions in the list that are supported by the IM Server.

A zero-length string is not a valid extension name. The IM library should disregard any zero-length strings that are returned in the extension list. The IM library does not use the requests which are not supported by the IM Server.

Setting IM Values

`XIM_SET_IM_VALUES` requests to set attributes to the IM.

```
XIM_SET_IM_VALUES (IM library -> IM Server)
     2     CARD16                  input-method-ID
     2     n                       byte length of im-attribute
```

```
     n     LISTofXIMATTRIBUTE     im-attributes
```

The im-attributes in `XIM_SET_IM_VALUES` message are specified as a LISTofXIMATTRIBUTE, specifying the attributes to be set. Attributes other than the ones returned by `XIM_OPEN_REPLY` message should not be specified.

`XIM_SET_IM_VALUES` is a synchronous request. The IM library should wait until receiving either an `XIM_SET_IM_VALUES_REPLY` packet or an `XIM_ERROR` packet, because it must receive the error attribute if `XIM_ERROR` message is returned.

```
XIM_SET_IM_VALUES_REPLY (IM Server -> IM library)
     2     CARD16     input-method-ID
     2                unused
```

`XIM_SET_IM_VALUES_REPLY` message returns the input-method-ID to distinguish replies from multiple IMs.

Getting IM Values

`XIM_GET_IM_VALUES` requests to query IM values supported by the IM Server currently being connected.

```
XIM_GET_IM_VALUES (IM library -> IM Server)
     2     CARD16           input-method-ID
     2     n                byte length of im-attribute-id
     n     LISTofCARD16     im-attribute-id
     p                      unused, p=Pad(n)
```

`XIM_GET_IM_VALUES` is a synchronous request. The IM library should wait until it receives either an `XIM_GET_IM_VALUES_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_GET_IM_VALUES_REPLY (IM Server -> IM library)
     2     CARD16                 input-method-ID
     2     n                      byte length of im-attributes returned
     n     LISTofXIMATTRIBUTE     im-attributes returned
```

The IM Server returns IM values with `XIM_GET_IM_VALUES_REPLY` message. The order of the returned im-attribute values corresponds directly to that of the list passed with the `XIM_GET_IM_VALUES` message.

Creating an IC

`XIM_CREATE_IC` message requests to create an IC.

```
XIM_CREATE_IC (IM library -> IM Server)
     2     CARD16                 input-method-ID
     2     n                      byte length of ic-attributes
     n     LISTofXICATTRIBUTE     ic-attributes
```

The input-context-id is specified by the IM Server to identify the client (IC). (It is not specified by the client in `XIM_CREATE_IC` message.), and it should not be set to zero.

`XIM_CREATE_IC` is a synchronous request which returns the input-context-ID. The IM library should wait until it receives either an `XIM_CREATE_IC_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_CREATE_IC_REPLY (IM Server -> IM library)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

Destroying the IC

`XIM_DESTROY_IC` message requests to destroy the IC.

```
XIM_DESTROY_IC (IM library -> IM Server)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

`XIM_DESTROY_IC` is a synchronous request. The IM library should not free its resources until it receives an `XIM_DESTROY_IC_REPLY` message because `XIM_DESTROY_IC` message may result in Callback packets such as `XIM_PREEDIT_DRAW` and `XIM_PREEDIT_DONE`.

```
XIM_DESTROY_IC_REPLY (IM Server -> IM library)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

Setting IC Values

`XIM_SET_IC_VALUES` messages requests to set attributes to the IC.

```
XIM_SET_IC_VALUES (IM library -> IM Server)
     2     CARD16                 input-method-ID
     2     CARD16                 input-context-ID
     2     n                      byte length of ic-attributes
     2                            unused
     n     LISTofXICATTRIBUTE     ic-attributes
```

The ic-attributes in `XIM_SET_IC_VALUES` message are specified as a LISTofXICATTRIBUTE, specifying the attributes to be set. Attributes other than the ones returned by `XIM_OPEN_REPLY` message should not be specified.

`XIM_SET_IC_VALUES` is a synchronous request. The IM library should wait until receiving either an `XIM_SET_IC_VALUES_REPLY` packet or an `XIM_ERROR` packet, because it must receive the error attribute if `XIM_ERROR` message is returned.

```
XIM_SET_IC_VALUES_REPLY (IM Server -> IM library)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

Getting IC Values

`XIM_GET_IC_VALUES` message requests to query IC values supported by the IM Server currently being connected.

```
XIM_GET_IC_VALUES (IM library -> IM Server)

     2     CARD16           input-method-ID
     2     CARD16           input-context-ID
     2     n                byte length of ic-attribute-id
     n     LISTofCARD16     ic-attribute-id
     p                      unused, p=Pad(2+n)
```

In LISTofCARD16, the appearance of the ic-attribute-id for the separator of NestedList shows the end of the heading nested list.

`XIM_GET_IC_VALUES` is a synchronous request and returns each attribute with its values to show the correspondence. The IM library should wait until receiving either an `XIM_GET_IC_VALUES_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_GET_IC_VALUES_REPLY (IM Server -> IM library)
     2     CARD16                 input-method-ID
     2     CARD16                 input-context-ID
     2     n                      byte length of ic-attribute
     2                            unused
     n     LISTofXICATTRIBUTE     ic-attribute
```

Setting IC Focus

`XIM_SET_IC_FOCUS` message requests to set the focus to the IC.

```
XIM_SET_IC_FOCUS (IM library -> IM Server)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

`XIM_SET_IC_FOCUS` is an asynchronous request.

Unsetting IC Focus

`XIM_UNSET_IC_FOCUS` message requests to unset the focus to the focused IC.

```
XIM_UNSET_IC_FOCUS (IM library -> IM Server)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

`XIM_UNSET_IC_FOCUS` is an asynchronous request.

Filtering Events

Event filtering is mainly provided for BackEnd method to allow input method to capture X events transparently to clients.

X Events are forwarded by `XIM_FORWARD_EVENT` message. This message can be operated both synchronously and asynchronously. If the requester sets the synchronous flag, the receiver must send `XIM_SYNC_REPLY` message back to the requester when all the data processing is done.

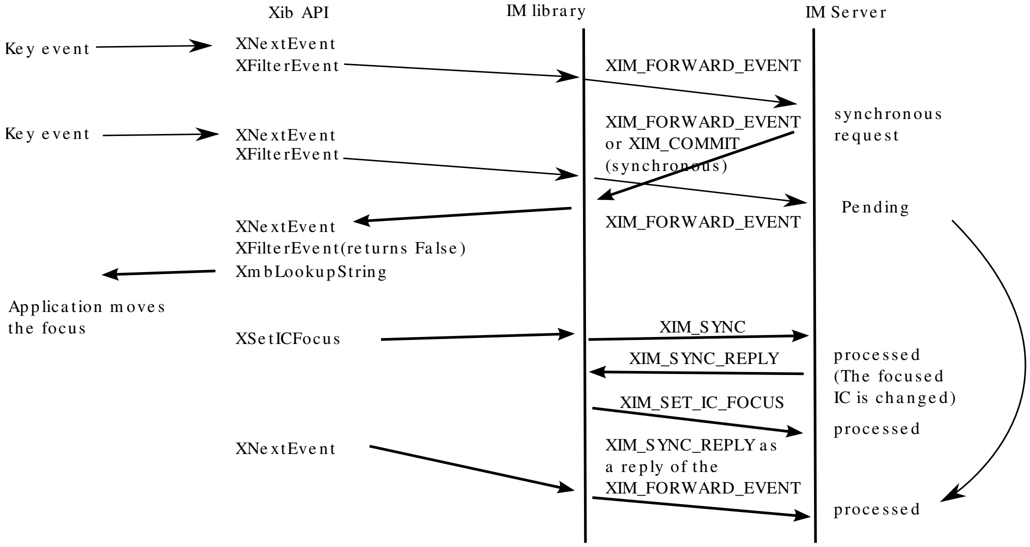
Protocol flow of BackEnd model

With BackEnd method, the protocol flow can be classified into two methods in terms of synchronization, depending on the synchronous-eventmask of `XIM_SET_EVENT_MASK` message. One can be called on-demand-synchronous method and another can be called as full-synchronous method.

In on-demand-synchronous method, the IM library always receives `XIM_FORWARD_EVENT` or `XIM_COMMIT` message as a synchronous request. Also, the IM Server needs to synchronously process the correspondent reply from the IM library and the following `XIM_FORWARD_EVENT` message sent from the IM library when any of the event causes the IM Server to send `XIM_FORWARD_EVENT` or `XIM_COMMIT` message to the IM library, so that the input service is consistent. If the IM library gets the control back from the application after receiving the synchronous request, the IM library replies for the synchronous request before processing any of the events. In this time, the IM Server blocks `XIM_FORWARD_EVENT` message which is sent by the IM library, and handles it after receiving the reply. However, the IM Server handles the other protocols at any time.

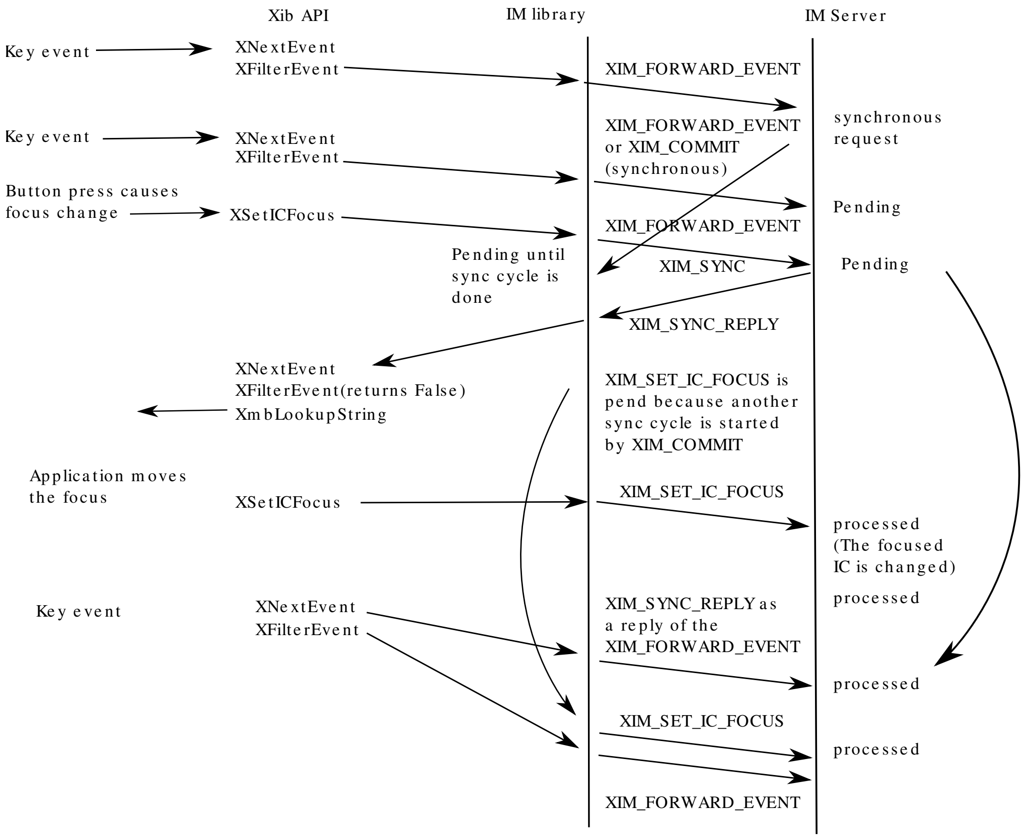
In full-synchronous method, the IM library always sends `XIM_FORWARD_EVENT` message to the IM Server as a synchronous request. Therefore, the reply to it from the IM Server will be put between the `XIM_FORWARD_EVENT` message and its `XIM_SYNC_REPLY` message. In case of sending `XIM_FORWARD_EVENT` or `XIM_COMMIT` message, the IM Server should set the synchronous flag off. Because the synchronization can be done by the following `XIM_SYNC_REPLY` message.

Following chart shows one of the simplest protocol flow which only deals with keyevents for preediting operation.



Sample Protocol Flow

Following chart shows one of the complex protocol flow, which deals with multiple focus windows and button press event as well as keyevent, and the focus is moved by the application triggered by both of keyevent and button press event.



Sample Protocol Flow 2

```
XIM_FORWARD_EVENT (IM library <--> IM Server)
     2     CARD16         input-method-ID
     2     CARD16         input-context-ID
     2     BITMASK16      flag
           #0001          synchronous
           #0002          request filtering (*1)
           #0004          request lookupstring (*2)
     2     CARD16         serial number
           XEVENT         X event
```

```
(*1)  Indicate the receiver should filter events and possible preedit may be invok
```

```
(*2)  Indicate the receiver should only do lookup string. The IM Server is expecte
to just do a conversion of the key event to the best candidate. This bit may
affect the state of the preedit state (e.g. compose of dead key sequences).
```

XEVENT format is same as the X Protocol event format(xEvent). As the value of xEvent's sequenceNumber is the bottom of 16 bit of XEvent's xany.serial, the top of 16 bit is sent by serial number(INT16).

`XIM_FORWARD_EVENT` message is used for forwarding the events from the IM library to the IM Server in order for IM to be able to filter the event. On the other hand, this message is also used for forwarding the events from the IM Server to the IM library if the event forwarded from the IM library is not filtered. The IM Server, which receives `XIM_FORWARD_EVENT` message without synchronous bit, should set synchronous bit. If both "request event filtering" and "request lookupstring" flag are set, then both filtering and lookup should be done for the same event.

Synchronizing with the IM Server

`XIM_SYNC` message requests to synchronize the IM library and the IM Server.

```
XIM_SYNC (IM library <--> IM Server)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

This synchronization can be started either on the IM library side or on the IM Server side. The side which receives `XIM_SYNC` message should process all XIM requests before replying. The input-context-ID is necessary to distinguish the IC with which the IM library and the IM Server are synchronized.

```
XIM_SYNC_REPLY (IM Server <--> IM library)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

The side which receives `XIM_FORWARD_EVENT`, `XIM_COMMIT` or any other message with synchronous bit, should process all XIM request before replying, and send `XIM_SYNC_REPLY` message as the reply to the previous message.

Sending a committed string

When the IM Server commits a string, the IM Server sends either the committed string or list of KeySym, or both, by `XIM_COMMIT` message.

```
XIM_COMMIT (IM Server -> IM library)

     2     CARD16          input-method-ID
     2     CARD16          input-context-ID
     2     BITMASK16       flag
           #0001           synchronous
           #0002           XLookupChars
           #0004           XLookupKeySym
           #0006           XLookupBoth = XLookupChars | XLookupKeySym
```

If flag is XLookupKeySym, the arguments continue as follows:

```
     2                unused
     4     KEYSYM     KeySym
```

If flag is XLookupChars, the arguments continue as follows

```
     2     m              byte length of committed string
     m     LISTofBYTE     committed string
     p                    unused, p = Pad(m)
```

If flag is XLookupBoth, the arguments continue as follows

```
     2                      unused
     4     KEYSYM           KeySym
     2     n                byte length of committed string
     n     LISTofBYTE       committed string
     p                      unused, p = Pad(2+n)
```

The IM Server which receives `XIM_COMMIT` message without synchronous bit should set synchronous bit.

Reset IC

`XIM_RESET_IC` message requests to reset the status of IC in the IM Server.

```
XIM_RESET_IC (IM library -> IM Server)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

`XIM_RESET_IC` is a synchronous request. The IM library should wait until receiving either an `XIM_RESET_IC_REPLY` packet or an `XIM_ERROR` packet.

```
XIM_RESET_IC_REPLY (IM Server -> IM library)

     2     CARD16           input-method-ID
     2     CARD16           input-context-ID
     2     n                byte length of preedit string
     n     LISTofBYTE       preedit string
     p                      unused, p = Pad(2+n)
```

`XIM_RESET_IC_REPLY` message returns the input-context-ID to distinguish replies from multiple ICs.

Callbacks

If XIMStyle has XIMPreeditArea or XIMStatusArea set, XIMGeometryCallback may be used, and if XIMPreeditCallback and/or XIMStatusCallback are set, corresponding callbacks may be used.

Any callback request may be sent from an IM Server to an IM client asynchronously in response to any request previously sent by the IM client to the IM Server.

When an IM Server needs to send a callback request synchronously with the request previously sent by an IM client, the IM Server sends it before replying to the previous request.

Negotiating geometry

The IM Server sends `XIM_GEOMETRY` message to start geometry negotiation, if XIMStyle has XIMPreeditArea or XIMStatusArea set.

```
XIM_GEOMETRY (IM Server -> IM library)
```

```
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

There is always a single Focus Window, even if some input fields have only one IC.

Converting a string

```
XIM_STR_CONVERSION (IM Server -> IM library)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
     2     CARD16     XIMStringConversionPosition
     2                unused
     4     CARD32     XIMCaretDirection
           #0         XIMForwardChar
           #1         XIMBackwardChar
           #2         XIMForwardWord
           #3         XIMBackwardWord
           #4         XIMCaretUp
           #5         XIMCaretDown
           #6         XIMNextLine
           #7         XIMCPreviousLine
           #8         XIMLineStart
           #9         XIMLineEnd
           #10        XIMAbsolutePosition
           #11        XIMDontChange
     2     CARD16     factor
     2     CARD16     XIMStringConversionOperation
           #0001      XIMStringConversionSubstitution
           #0002      XIMStringConversionRetrieval
     2     INT16      byte length to multiply the XIMStringConversionType
```

`XIM_STR_CONVERSION` message may be used to start the string conversion from the IM Server.

```
XIM_STR_CONVERSION_REPLY (IM library -> IM Server)

     2     CARD16             input-method-ID
     2     CARD16             input-context-ID
     4     CARD32             XIMStringConversionFeedback
           XIMSTRCONVTEXT     XIMStringConversionText
```

`XIM_STR_CONVERSION_REPLY` message returns the string to be converted and the feedback information array.

Preedit Callbacks

The IM Server sends `XIM_PREEDIT_START` message to call the XIMPreeditStartCallback function.

```
XIM_PREEDIT_START (IM Server -> IM library)
```

2	CARD16	input-method-ID
2	CARD16	input-context-ID

The reply to this message must be sent synchronously. The reply forwards the return value from the callback function to the IM Server.

`XIM_PREEDIT_START_REPLY (IM library -> IM Server)`

2	CARD16	input-method-ID
2	CARD16	input-context-ID
4	INT32	return value

`XIM_PREEDIT_START_REPLY` message returns the input-context-ID to distinguish replies from multiple IC's. The return value contains the return value of the function XIMPreeditStartCallback.

The IM Server sends `XIM_PREEDIT_DRAW` message to call the XIMPreeditDrawCallback function.

`XIM_PREEDIT_DRAW (IM Server -> IM library)`

2	CARD16	input-method-ID
2	CARD16	input-context-ID
4	INT32	caret
4	INT32	chg_first
4	INT32	chg_length
4	BITMASK32	status
	#x0000001	no string
	#x0000002	no feedback
2	n	length of preedit string
n	STRING8	preedit string
p		unused, p = Pad(2+n)
2	m	byte length of feedback array
2		unused
m	LISTofXIMFEEDBACK	feedback array

The fields "caret", "chg_first" and "chg_length" correspond to the fields of XIMPreeditDrawCallbackStruct. When the "no string" bit of the status field is set, the text field of XIMPreeditDrawCallbackStruct is NULL. When the "no feedback" bit of the status field is set, the text feedback field of XIMPreeditDrawCallbackStruct is NULL. When the above bits are not set, "preedit string" contains the preedit string to be displayed, and the feedback array contains feedback information.

The IM Server sends `XIM_PREEDIT_CARET` message to call the PreeditCaretCallback function.

`XIM_PREEDIT_CARET (IM Server -> IM library)`

2	CARD16	input-method-ID
2	CARD16	input-context-ID
4	INT32	position
4	CARD32	direction
	#0	XIMForwardChar

```
            #1         XIMBackwardChar
            #2         XIMForwardWord
            #3         XIMBackwardWord
            #4         XIMCaretUp
            #5         XIMCaretDown
            #6         XIMNextLine
            #7         XIMCPreviousLine
            #8         XIMLineStart
            #9         XIMLineEnd
            #10        XIMAbsolutePosition
            #11        XIMDontChange
      4     CARD32     style
            #0         XIMInvisible
            #1         XIMCPrimary
            #2         XIMSecondary
```

Each entry corresponds to a field of XIMPreeditCaretCallbackStruct. Since this callback sets the caret position, its reply must be sent synchronously.

```
XIM_PREEDIT_CARET_REPLY (IM library -> IM Server)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
     4     CARD32     position
```

The position is the value returned by the callback function after it has been called.

The IM Server sends `XIM_PREEDIT_DONE` message to call the XIMPreeditDoneCallback function.

```
XIM_PREEDIT_DONE (IM Server -> IM library)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

Preedit state notify

```
XIM_PREEDITSTATE (IM Server -> IM Library)
     2     CARD16        input-method-ID
     2     CARD16        input-context-ID
     4     BITMASK32     XIMPreeditState
           #x0000000     XIMPreeditUnknown
           #x0000001     XIMPreeditEnable
           #x0000002     XIMPreeditDisable
```

`XIM_PREEDITSTATE` message is used to call the XIMPreeditStateNotifyCallback function.

Status Callbacks

The IM Server sends `XIM_STATUS_START` message to call the XIMStatusStartCallback function.

```
XIM_STATUS_START (IM Server -> IM library)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

The IM Server sends `XIM_STATUS_DRAW` message to call the XIMStatusDrawCallback function.

```
XIM_STATUS_DRAW (IM Server -> IM library)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
     4     CARD32     type
           #0         XIMTextType
           #1         XIMBitmapType
```

If type is XIMTextType, the arguments continue as follows.

```
     4     BITMASK32            status
           #x0000001            no string
           #x0000002            no feedback
     2     n                    length of status string
     n     STRING8              status string
     p                          unused, p = Pad(2+n)
     2     m                    byte length of feedback array
     2                          unused
     m     LISTofXIMFEEDBACK    feedback array
```

If type is XIMBitmapType, the arguments continue as follows.

```
     4     PIXMAP     pixmap data
```

The field "type" corresponds to the field in XIMStatusDrawCallbackStruct.

The IM Server sends `XIM_STATUS_DONE` message to call the XIMStatusDoneCallback function.

```
XIM_STATUS_DONE (IM Server -> IM library)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

Acknowledgements

This document represents the culmination of several years of debate and experiments done under the auspices of the MIT X Consortium i18n working group. Although this was a group effort, the author remains responsible for any errors or omissions.

We would like to thank to all members of this group. And we would like to make special thanks to the following people (in alphabetical order) for their participa-

tion in the IM Protocol design, Hector Chan, Takashi Fujiwara, Yoshio Horiuchi, Makoto Inada, Hiromu Inukai, Mickael Kung, Seiji Kuwari, Franky Ling, Hiroyuki Machida, Hiroyuki Miyamoto, Frank Rojas, Bob Scheifler, Makiko Shimamura, Shoji Sugiyama, Hidetoshi Tajima, Masaki Takeuchi, Makoto Wakamatsu, Masaki Wakao, Nobuyuki Tanaka, Shigeru Yamada, Katsuhisa Yano, Jinsoo Yoon.

References

X Window System Protocol Version 11. Robert W. Scheifler.

Xlib - C Language X Interface". Robert W. Scheifler.

A. Common Extensions

Extension opcodes and packet names (e.g. `XIM_EXT_SET_EVENT_MASK`) for additional extensions may be registered with X Consortium. The following is a commonly well-known extended packet.

(1) Extension to manipulate the event handling\fP

`XIM_EXT_SET_EVENT_MASK` message specifies the set of event masks that the IM library should manipulate.

```
XIM_EXT_SET_EVENT_MASK (IM Server -> IM library)

     2     CARD16        input-method-ID
     2     CARD16        input-context-ID
     4     EVENTMASK     filter-event-mask (*1)
     4     EVENTMASK     intercept-event-mask (*2)
     4     EVENTMASK     select-event-mask (*3)
     4     EVENTMASK     forward-event-mask (*4)
     4     EVENTMASK     synchronous-event-mask (*5)

     (*1) Specify the events to be neglected by the IM library via XFilterEvent.
     (*2) Specify the events to be deselected by the IM library with XSelectInput.
     (*3) Specify the events to be selected by the IM library with XSelectInput.
     (*4) Specify all the events to be forwarded to the IM Server by the IM librar
     (*5) Specify the events to be forwarded with synchronous flag on by the IM li
```

The IM library must reply `XIM_SYNC_REPLY` message to the IM Server. This request is valid after the ic is created.

(2) Extension for improvement of performance.

The following requests may be used for improvement of performance.

`XIM_EXT_FORWARD_KEYEVENT` message may be used instead of `XIM_FORWARD_EVENT` message.

```
XIM_EXT_FORWARD_KEYEVENT (IM Server <--> IM library)
     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
```

```
     2     BITMASK16     flag
          #0001     synchronous
     2     CARD16     sequence number
     1     BYTE     xEvent.u.u.type
     1     BYTE     keycode
     2     CARD16     state
     4     CARD32     time
     4     CARD32     window
```

`XIM_EXT_MOVE` message may be used to change the spot location instead of XIM_SET_IC_VALUES message. It is effective only if the client specified XIMPreeditPosition.

```
XIM_EXT_MOVE (IM library -> IM Server)

     2     CARD16     input-method-ID
     2     CARD16     input-context-ID
     2     INT16     X
     2     INT16     Y
```

`XIM_EXT_MOVE` message is a asynchronous request.

B. Transport List

The list of transport specific IM Server address format registered

The following format represents the ATOM contained in `XIM_SERVERS` property and the string returned from the request converting selection target LOCALES and TRANSPORT.

```
     "{category=[value,...]}..."
```

The following categories are currently registered.

```
server;: IM Server name (used for XIM_SERVERS)
locale;: XPG4 locale name (LOCALES)
transport;: transport-specific name (TRANSPORT)
```

The preregistered formats for transport-specific names are as follows:

`TCP/IP Names`

The following syntax should be used for system internal domain names:

```
<local name>  ::= "local/"<hostname>":"<pathname>
```

Where <*pathname*> is a path name of socket address.

IM Server's name should be set to <*pathname*> to run multiple IM Server at the same time

The following syntax should be used for Internet domain names:

```
<TCP name>  ::=  "tcp/"<hostname>":"<ipportnumber>
```

where *<hostname>* is either symbolic (such as expo.lcs.mit.edu) or numeric decimal (such as 18.30.0.212). The *<ipportnumber>* is the port on which the IM Server is listening for connections. For example:

```
tcp/expo.lcs.mit.edu:8012
tcp/18.30.0.212:7890
```

`DECnet Names`

The following syntax should be used for DECnet names:

```
<DECnet name>  ::=  "decnet/"<nodename>"::IMSERVER$"<objname>
```

where *<nodename>* is either symbolic (such as SRVNOD) or the numeric decimal form of the DECnet address (such as 44.70). The *<objname>* is normal, case-insensitive DECnet object name. For example:

```
DECNET/SRVNOD::IMSERVER$DEFAULT
decnet/44.70::IMSERVER$other
```

`X Names`

The following syntax should be used for X names:

```
<X name>  ::=  "X/"
```

If a given category has multiple values, the value is evaluated in order of setting.

C. Protocol Number

`Major Protocol number`

```
XIM_CONNECT                                  #001
XIM_CONNECT_REPLY                            #002
XIM_DISCONNECT                               #003
XIM_DISCONNECT_REPLY                         #004

XIM_AUTH_REQUIRED                            #010
XIM_AUTH_REPLY                               #011
XIM_AUTH_NEXT                                #012
XIM_AUTH_SETUP                               #013
XIM_AUTH_NG                                  #014

XIM_ERROR                                    #020

XIM_OPEN                                     #030
XIM_OPEN_REPLY                               #031
XIM_CLOSE                                    #032
```

XIM_CLOSE_REPLY	#033
XIM_REGISTER_TRIGGERKEYS	#034
XIM_TRIGGER_NOTIFY	#035
XIM_TRIGGER_NOTIFY_REPLY	#036
XIM_SET_EVENT_MASK	#037
XIM_ENCODING_NEGOTIATION	#038
XIM_ENCODING_NEGOTIATION_REPLY	#039
XIM_QUERY_EXTENSION	#040
XIM_QUERY_EXTENSION_REPLY	#041
XIM_SET_IM_VALUES	#042
XIM_SET_IM_VALUES_REPLY	#043
XIM_GET_IM_VALUES	#044
XIM_GET_IM_VALUES_REPLY	#045

XIM_CREATE_IC	#050
XIM_CREATE_IC_REPLY	#051
XIM_DESTROY_IC	#052
XIM_DESTROY_IC_REPLY	#053
XIM_SET_IC_VALUES	#054
XIM_SET_IC_VALUES_REPLY	#055
XIM_GET_IC_VALUES	#056
XIM_GET_IC_VALUES_REPLY	#057
XIM_SET_IC_FOCUS	#058
XIM_UNSET_IC_FOCUS	#059
XIM_FORWARD_EVENT	#060
XIM_SYNC	#061
XIM_SYNC_REPLY	#062
XIM_COMMIT	#063
XIM_RESET_IC	#064
XIM_RESET_IC_REPLY	#065

XIM_GEOMETRY	#070
XIM_STR_CONVERSION	#071
XIM_STR_CONVERSION_REPLY	#072
XIM_PREEDIT_START	#073
XIM_PREEDIT_START_REPLY	#074
XIM_PREEDIT_DRAW	#075
XIM_PREEDIT_CARET	#076
XIM_PREEDIT_CARET_REPLY	#077
XIM_PREEDIT_DONE	#078
XIM_STATUS_START	#079
XIM_STATUS_DRAW	#080
XIM_STATUS_DONE	#081
XIM_PREEDITSTATE	#082

`(*) The IM Server's extension protocol number should be more than #128.`

D. Implementation Tips

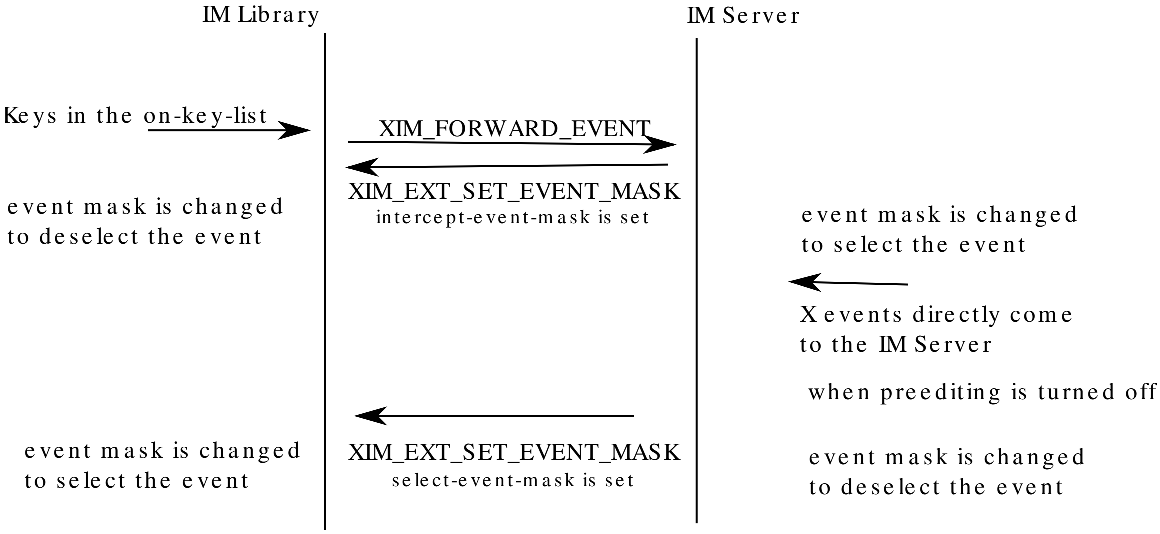
(1) FrontEnd Method

FrontEnd method is recognized as a performance acceleration by the trade off of the variety of the reliability.

In order to use the FrontEnd method, the IM library must query the IM Server to see if the FrontEnd extension is available. The query is made by using the `XIM_QUERY_EXTENSION` message. The IM Server may send `XIM_EXT_SET_EVENT_MASK` message with intercept-event-mask, forward-event-mask, and synchronous-event-mask values set after replying `XIM_QUERY_EXTENSION_REPLY` message.

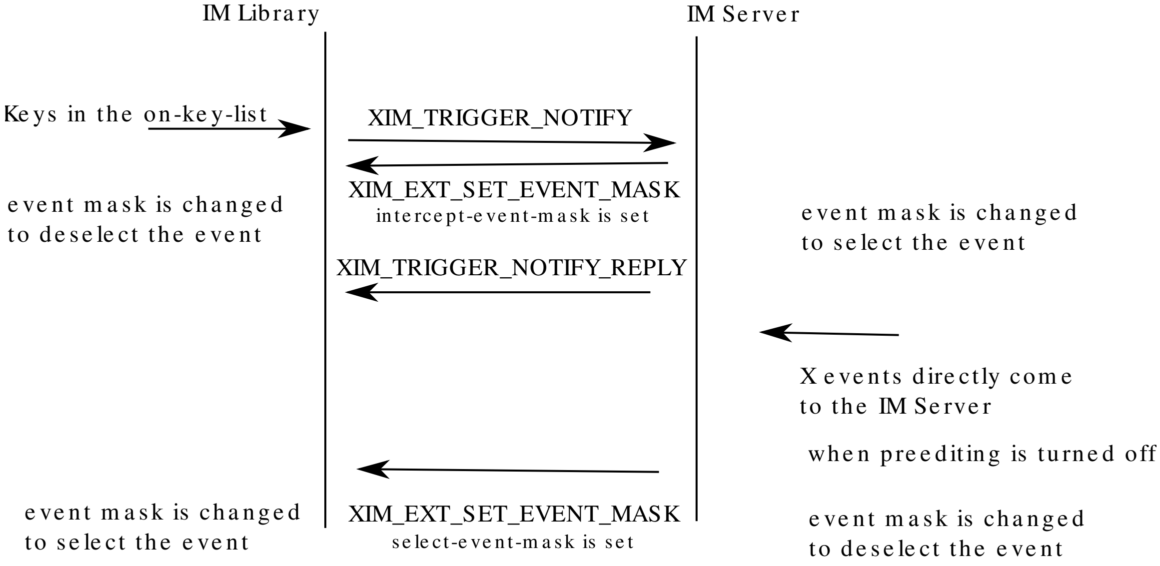
FrontEnd method can be implemented in a couple of ways depending on how the IM Server utilize `XIM_EXT_SET_EVENT_MASK` message.

One approach is to update both of the input mask and the filter-event-mask depending on the preeidting state. The sample protocol sequence using the static event flow is as follows:



The flow of events

To pursuit a maximum performance regardless of the preediting mode, the IM Server may use the dynamic event flow with the following sample protocol sequence.



The flow of events

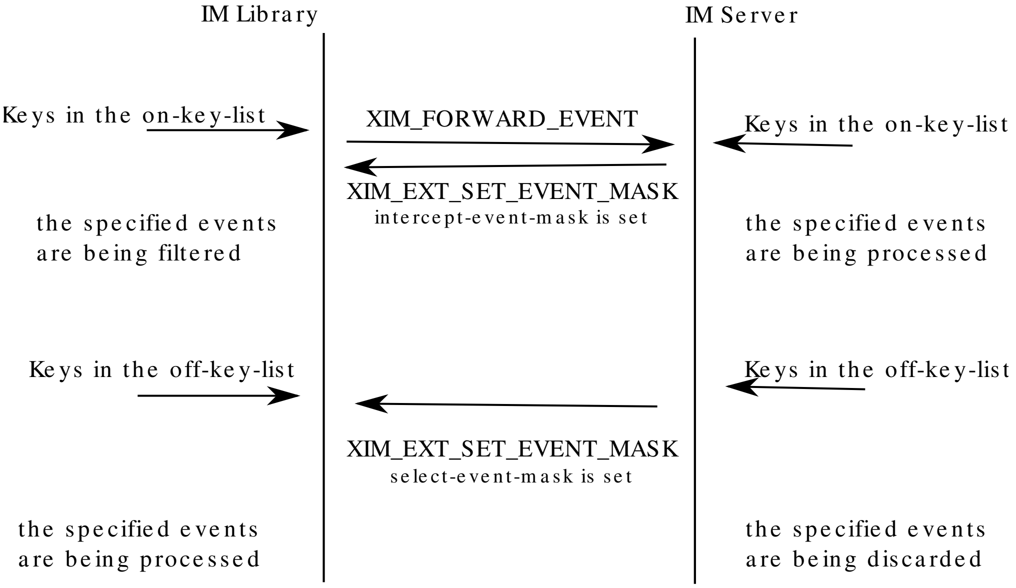
This method can reduce the XIM protocol traffic dramatically by updating intercept-event-mask and select-event-mask accordingly. The tradeoff of this performance improvement is that the key events may be lost or disordered in some particular situation, such as when the user types the keyboard in following sequence really fast:

<preediting on key>"some strings"<preediting off key>"another string"

Since this method requires the input mask updates to the both the IM Server and Xlib when turning on and off the preediting, and there is a time lag till the requests take effect when two client issues the input mask updates simultaneously.

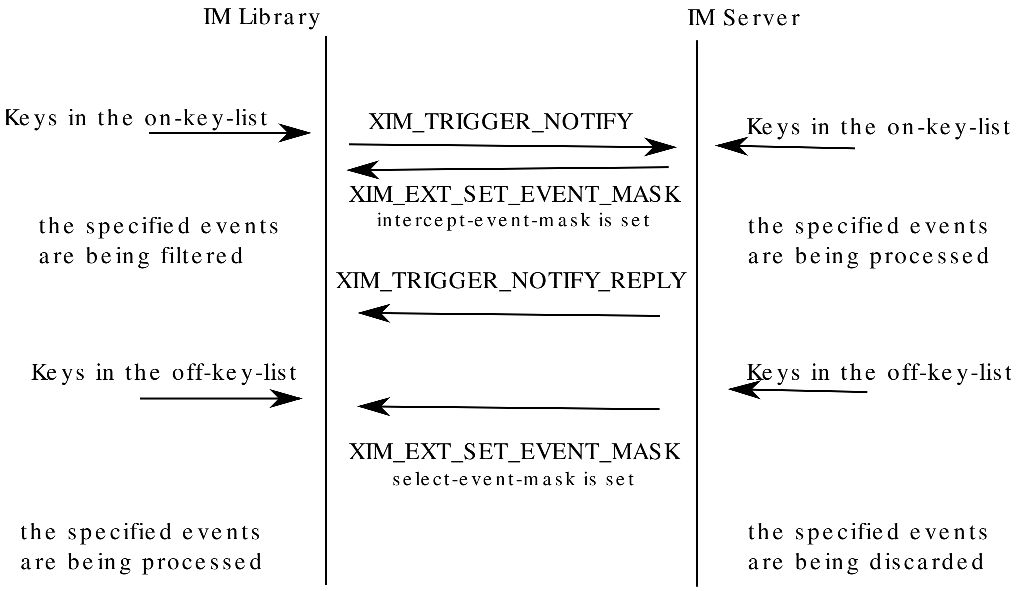
Another approach of the FrontEnd method is to update the filter-event-mask depending on the preediting state and not to update the input mask. The IM Server must register both of the preediting on key list and off key list by `XIM_REGISTER_TRIGGERKEYS` message. In this method, Both the IM Server and the IM client select the same events on the same client's window, so that the events are delivered to both of the IM Server and the client. The preediting on and off states are expressed by whether the key events are filtered or not. The sample protocol sequence are as follows:

<<Using static event flow>>



The flow of events

<<Using the dynamic event flow>>



The flow of events

This method does not have the problem of the time lag when going across the preediting on and off mode, however, the amount of the performance acceleration is not as good as the method described above.

In general, the FrontEnd method requires some synchronization to some of the X protocols, such as the ChangeWindowAttribute protocol for the event mask change or the GrabKey protocol, since it relies on the X's principal event dispatching mechanism. Any X protocol bindings do not consider the synchronization might cause some mis-synchronization between the IM clients and the IM Server.

(2) Transport Layer

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.

The interface layer and the transport layer make various communication channels usable such as X Protocol, TCP/IP, DECnet or STREAM. The following is a sample implementation for the transporter using the X connection. Refer to "xtrans" for the transporter using Socket 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".

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 D.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 D.2. The ClientMessage sent by the 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 D.3. The read/write method and the major/minor-transport-version

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

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

read/write

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

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 D.4. 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	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 D.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.

- 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.
- 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 D.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

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 D.7. The ClientMessage event's format to send Atom of property

Structure Members		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

Format for the data from the IM Server to the Client

ClientMessage

The format of the ClientMessage is as follows:

Table D.8. The ClientMessage event's format (first or middle)

Structure Members		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 D.9. The ClientMessage event's format (only or last)

Structure Members		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.

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

- 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 D.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 D.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

Closing Connection

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