Record Extension Protocol Specification

X Consortium Standard

Martha Zimet, Network Computing Devices, Inc.
Edited by Stephen Gildea

Record Extension Protocol Specification: X Consortium Standard

by Martha Zimet and Stephen Gildea

X Version 11, Release 7.7

Copyright © 1994 Network Computing Devices, Inc.

Permission to use, copy, modify, distribute, and sell this documentation for any purpose is hereby granted without fee, provided that the above copyright notice and this permission notice appear in all copies. Network Computing Devices, Inc. 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, 1995 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 and 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.

Table of Contents

1. Introduction ... 1
    Acknowledgements ... 2
    Goals ... 2
    Requirements ... 2
2. Design ... 3
    Overview ... 3
        Data Delivery ... 3
        Record Context ... 3
        Record Client Connections ... 4
        Events ... 4
        Timing ... 4
    Types ... 4
    Errors ... 7
3. Protocol Requests ... 8
4. Encoding ... 14
    Types ... 14
    Errors ... 15
    Requests ... 15

Chapter 1. Introduction

Several proposals have been written over the past few years that address some of the issues surrounding the recording and playback of user actions in the X Window System[1] :

- *Some Proposals for a Minimal X11 Testing Extension*, Kieron Drake, UniSoft Ltd., April 1991
- *X11 Input Synthesis Extension Proposal*, Larry Woestman, Hewlett Packard, November 1991
- *XTrap Architecture*, Dick Annicchiario, et al, Digital Equipment Corporation, July 1991
- *XTest Extension Recording Specification*, Yochanan Slonim, Mercury Interactive, December 1992

This document both unifies and extends the previous diverse approaches to generate a proposal for an X extension that provides support for the recording of all core X protocol and arbitrary extension protocol. Input synthesis, or playback, has already been implemented in the XTest extension, an X Consortium standard. Therefore, this extension is limited to recording.

In order to provide both record and playback functionality, a hypothetical record application could use this extension to capture both user actions and their consequences. For example, a button press (a user action) may cause a window to be mapped and a corresponding `MapNotify` event to be sent (a consequence). This information could be stored for later use by a playback application.

The playback application could use the recorded actions as input for the XTest extension's `XTestFakeInput` operation to synthesize the appropriate input events. The "consequence" or synchronization information is then used as a synchronization point during playback. That is, the playback application does not generate specific synthesized events until their matching synchronization condition occurs. When the condition occurs the processing of synthesized events continues. Determination that the condition has occurred may be made by capturing the consequences of the synthesized events and comparing them to the previously recorded synchronization information. For example, if a button press was followed by a `MapNotify` event on a particular window in the recorded data, the playback application might synthesize the button press then wait for the `MapNotify` event on the appropriate window before proceeding with subsequent synthesized input.

Because it is impossible to predict what synchronization information will be required by a particular application, the extension provides facilities to record any subset of core X protocol and arbitrary extension protocol. As such, this extension does not enforce a specific synchronization methodology; any method based on information in the X protocol stream (e.g., watching for window mapping/unmapping, cursor changes, drawing of certain text strings, etc.) can capture the information it needs using RECORD facilities.

---

[1] *X Window System* is a trademark of The Open Group.

Acknowledgements

The document represents the culmination of two years of debate and experiments done under the auspices of the X Consortium xtest working group. Although this was a group effort, the author remains responsible for any errors or omissions. Two years ago, Robert Chesler of Absol-puter, Kieron Drake of UniSoft Ltd., Marc Evans of Synergytics and Ken Miller of Digitial shared the vision of a standard extension for recording and were all instrumental in the early protocol development. During the last two years, Bob Scheifler of the X Consortium and Jim Fulton of NCD continuously provided input to the protocol design, as well as encouragement to the author. In the last few months, Stephen Gildea and Dave Wiggins, both X Consortium staff, have spent considerable time fine tuning the protocol design and reviewing the protocol specifications. Most recently, Amnon Cohen of Mercury Interactive has assisted in clarification of the recorded event policy, and Kent Siefkes of Performance Awareness has assisted in clarification of the timestamp policy.

Goals

- To provide a standard for recording, whereby both device events and synchronization information in the form of device event consequences are recorded.
- To record contextual information used in synchronized playback without prior knowledge of the application that is being recorded.
- To provide the ability to record arbitrary X protocol extensions.

Requirements

The extension should function as follows:

- It should not be dependent on other clients or extensions for its operation.
- It should not significantly impact performance.
- It should support the recording of all device input (core devices and XInput devices).
- It should be extendible.
- It should support the recording of synchronization information for user events.

Chapter 2. Design

This section gives an overview of the RECORD extension and discusses its overall operation and data types.

Overview

The mechanism used by this extension for recording is to intercept core X protocol and arbitrary X extension protocol entirely within the X server itself. When the extension has been requested to intercept specific protocol by one or more clients, the protocol data are formatted and returned to the recording clients.

The extension provides a mechanism for capturing all events, including input device events that go to no clients, that is analogous to a client expressing "interest" in all events in all windows, including the root window. Event filtering in the extension provides a mechanism for feeding device events to recording clients; it does not provide a mechanism for in-place, synchronous event substitution, modification, or withholding. In addition, the extension does not provide data compression before intercepted protocol is returned to the recording clients.

Data Delivery

Because events are limited in size to 32 bytes, using events to return intercepted protocol data to recording clients is prohibitive in terms of performance. Therefore, intercepted protocol data are returned to recording clients through multiple replies to the extension request to begin protocol interception and reporting. This utilization is consistent with `ListFontsWithInfo` , for example, where a single request has multiple replies.

Individual requests, replies, events or errors intercepted by the extension on behalf of recording clients cannot be split across reply packets. In order to reduce overhead, multiple intercepted requests, replies, events and errors might be collected into a single reply. Nevertheless, all data are returned to the client in a timely manner.

Record Context

The extension adds a record context resource (RC) to the set of resources managed by the server. All the extension operations take an RC as an argument. Although the protocol permits sharing of RCs between clients, it is expected that clients will use their own RCs. The attributes used in extension operations are stored in the RCs, and these attributes include the protocol and clients to intercept.

The terms "register" and "unregister" are used to describe the relationship between clients to intercept and the RC. To register a client with an RC means the client is added to the list of clients to intercept; to unregister a client means the client is deleted from the list of clients to intercept. When the server is requested to register or unregister clients from an RC, it is required to do so immediately. That is, it is not permissible for the server to wait until recording is enabled to register clients or recording is disabled to unregister clients.

Record Client Connections

The typical communication model for a recording client is to open two connections to the server and use one for RC control and the other for reading protocol data.

The "control" connection can execute requests to obtain information about the supported protocol version, create and destroy RCs, specify protocol types to intercept and clients to be recorded, query the current state of an RC, and to stop interception and reporting of protocol data. The "data" connection can execute a request to enable interception and reporting of specified protocol for a particular RC. When the "enable" request is issued, intercepted protocol is sent back on the same connection, generally in more than one reply packet. Until the last reply to the "enable" request is sent by the server, signifying that the request execution is complete, no other requests will be executed by the server on that connection. That is, the connection that data are being reported on cannot issue the "disable" request until the last reply to the "enable" request is sent by the server. Therefore, unless a recording client never has the need to disable the interception and reporting of protocol data, two client connections are necessary.

Events

The terms "delivered events" and "device events" are used to describe the two event classes recording clients may select for interception. These event classes are handled differently by the extension. Delivered events are core X events or X extension events the server actually delivers to one or more clients. Device events are events generated by core X devices or extension input devices that the server may or may not deliver to any clients. When device events are selected for interception by a recording client, the extension guarantees each device event is recorded and will be forwarded to the recording client in the same order it is generated by the device.

The recording of selected device events is not affected by server grabs. Delivered events, on the other hand, can be affected by server grabs. If a recording client selects both a device event and delivered events that result from that device event, the delivered events are recorded after the device event. In the absence of grabs, the delivered events for a device event precede later device events.

Requests that have side effects on devices, such as `WarpPointer` and `GrabPointer` with a confine-to window, will cause RECORD to record an associated device event. The XTEST extension request `XTestFakeInput` causes a device event to be recorded; the device events are recorded in the same order that the `XTestFakeInput` requests are received by the server.

If a key autorepeats, multiple `KeyPress` and `KeyRelease` device events are reported.

Timing

Requests are recorded just before they are executed; the time associated with a request is the server time when it is recorded.

Types

The following new types are used in the request definitions that appear in section 3.

RC: CARD32

The `"RC"` type is a resource identifier for a server record context.

RANGE8:	[first, last:	CARD8]
RANGE16:	[first, last:	CARD16]
EXTRANGE:	[major:	RANGE8
	minor:	RANGE16]

RECORDRANGE:	[core-requests:	RANGE8
	core-replies:	RANGE8
	ext-requests:	EXTRANGE
	ext-replies:	EXTRANGE
	delivered-events:	RANGE8
	device-events:	RANGE8
	errors:	RANGE8
	client-started:	BOOL
	client-died:	BOOL]

The `"RECORDRANGE"` structure contains the protocol values to intercept. Typically, this structure is sent by recording clients over the control connection when creating or modifying an RC.

- Specifies core X protocol requests with an opcode field between *first* and *last* inclusive. If *first* is equal to 0 and *last* is equal to 0, no core requests are specified by this RECORDRANGE. If *first* is greater than *last*, a `"Value"` error results.

- Specifies replies resulting from core X protocol requests with an opcode field between *first* and *last* inclusive. If *first* is equal to 0 and *last* is equal to 0, no core replies are specified by this RECORDRANGE. If *first* is greater than *last*, a `"Value"` error results.

- Specifies extension protocol requests with a major opcode field between *major.first* and *major.last* and a minor opcode field between *minor.first* and *minor.last* inclusive. If *major.first* and *major.last* are equal to 0, no extension protocol requests are specified by this RECORDRANGE. If *major.first* or *major.last* is less than 128 and greater than 0, if *major.first* is greater than *major.last*, or if *minor.first* is greater than *minor.last*, a `"Value"` error results.

- Specifies replies resulting from extension protocol requests with a major opcode field between *major.first* and *major.last* and a minor opcode field between *minor.first* and *minor.last* inclusive. If *major.first* and *major.last* are equal to 0, no extension protocol replies are specified by this RECORDRANGE. If *major.first* or *major.last* is less than 128 and greater than 0, if *major.first* is greater than *major.last*, or if *minor.first* is greater than *minor.last*, a `"Value"` error results.

- This is used for both core X protocol events and arbitrary extension events. Specifies events that are delivered to at least one client that have a code field between *first* and *last* inclusive. If *first* is equal to 0 and *last* is equal to 0, no events are specified by this RECORDRANGE. Otherwise, if *first* is less than 2 or *last* is less than 2, or if *first* is greater than *last*, a `"Value"` error results.

- This is used for both core X device events and X extension device events that may or may not be delivered to a client. Specifies device events that have a code field between *first* and *last* inclusive. If *first* is equal to 0 and *last* is equal to 0, no device events are specified by this RECORDRANGE. Otherwise, if *first* is less than 2 or *last* is less than 2, or if *first* is greater than *last*, a `"Value"` error results.

- Because the generated device event may or may not be associated with a client, unlike other RECORDRANGE components, which select protocol for a specific client, selecting for device events in any RECORDRANGE in an RC causes the recording client to receive one instance for each device event generated that is in the range specified.

- This is used for both core X protocol errors and arbitrary extension errors. Specifies errors that have a code field between *first* and *last* inclusive. If *first* is equal to 0 and *last* is equal to 0, no errors are specified by this RECORDRANGE. If *first* is greater than *last*, a `"Value"` error results.

- Specifies the connection setup reply. If `False` , the connection setup reply is not specified by this RECORDRANGE.

- Specifies notification when a client disconnects. If `False` , notification when a client disconnects is not specified by this RECORDRANGE.

ELEMENT_HEADER:	[from-server-time:	BOOL
	from-client-time:	BOOL
	from-client-sequence:	BOOL]

The `ELEMENT_HEADER` structure specifies additional data that precedes each protocol element in the *data* field of a `RecordEnableContext` reply.

- If *from-server-time* is `True` , each intercepted protocol element with category `FromServer` is preceded by the server time when the protocol was recorded.

- If *from-client-time* is `True` , each intercepted protocol element with category `FromClient` is preceded by the server time when the protocol was recorded.

- If *from-client-sequence* is `True` , each intercepted protocol element with category `FromClient` or `ClientDied` is preceded by the 32-bit sequence number of the recorded client's most recent request processed by the server at that time. For `FromClient` , this will be one less than the sequence number of the following request. For `ClientDied` , the sequence number will be the only data, because no protocol is recorded.

Note that a reply containing device events is treated the same as other replies with category `FromServer` for purposes of these flags. Protocol with category `FromServer` is never preceded by a sequence number because almost all such protocol has a sequence number in it anyway.

If both a server time and a sequence number have been requested for a reply, each protocol request is preceded first by the time and second by the sequence number.

XIDBASE: CARD32

The XIDBASE type is used to identify a particular client. Valid values are any existing resource identifier of any connected client, in which case the client that created the resource is specified, or the resource identifier base sent to the target client from the server in the connection setup reply. A value of 0 (zero) is valid when the XIDBASE is associated with device events that may not have been delivered to a client.

CLIENTSPEC: XIDBASE or {*CurrentClients*, *FutureClients*, *AllClients*}

The CLIENTSPEC type defines the set of clients the RC attributes are associated with. This type is used by recording clients when creating an RC or when changing RC attributes. XIDBASE specifies that the RC attributes apply to a single client only. `CurrentClients` specifies that the RC attributes apply to current client connections; `FutureClients` specifies future client connections; `AllClients` specifies all client connections, which includes current and future.

The numeric values for `CurrentClients` , `FutureClients` and `AllClients` are defined such that there will be no intersection with valid XIDBASEs.

When the context is enabled, the data connection is unregistered if it was registered. If the context is enabled, `CurrentClients` and `AllClients` silently exclude the recording data connection. It is an error to explicitly register the data connection.

CLIENT_INFO:	[client-resource:	CLIENTSPEC
	intercepted-protocol:	LISTofRECORDRANGE]

This structure specifies an intercepted client and the protocol to be intercepted for the client. The *client-resource* field is a resource base that identifies the intercepted client. The *intercepted-protocol* field specifies the protocol to intercept for the *client-resource*.

Errors

RecordContext

- This error is returned if the value for an RC argument in a request does not name a defined record context.

Chapter 3. Protocol Requests

`RecordQueryVersion`

- *major-version*, *minor-version*: CARD16

->

- *major-version*, *minor-version*: CARD16

This request specifies the RECORD extension protocol version the client would like to use. When the specified protocol version is supported by the extension, the protocol version the server expects from the client is returned. Clients must use this request before other RECORD extension requests.

This request also determines whether or not the RECORD extension protocol version specified by the client is supported by the extension. If the extension supports the version specified by the client, this version number should be returned. If the client has requested a higher version than is supported by the server, the server's highest version should be returned. Otherwise, if the client has requested a lower version than is supported by the server, the server's lowest version should be returned. This document defines major version one (1), minor version thirteen (13).

`RecordCreateContext`

context: RC

element-header: ELEMENT_HEADER

client-specifiers: LISTofCLIENTSPEC

ranges: LISTofRECORDRANGE

Errors: `Match` , `Value` , `IDChoice` , `Alloc`

This request creates a new record context within the server and assigns the identifier *context* to it. After the *context* is created, this request registers the set of clients in *client-specifiers* with the *context* and specifies the protocol to intercept for those clients. The recorded protocol elements will be preceded by data as specified by *element-header*. Typically, this request is used by a recording client over the control connection. Multiple RC objects can exist simultaneously, containing overlapping sets of protocol and clients to intercept.

If any of the values in *element-header* or *ranges* is invalid, a "`Value`" error results. Duplicate items in the list of *client-specifiers* are ignored. If any item in the *client-specifiers* list is not a valid CLIENTSPEC, a "`Match`" error results. Otherwise, each item in the *client-specifiers* list is processed as follows:

- If the item is an XIDBASE identifying a particular client, the specified client is registered with the *context* and the protocol to intercept for the client is then set to *ranges*.

- If the item is `CurrentClients` , all existing clients are registered with the *context* at this time. The protocol to intercept for all clients registered with the *context* is then set to *ranges*.

- If the item is `FutureClients`, all clients that connect to the server after this request executes will be automatically registered with the *context*. The protocol to intercept for such clients will be set to *ranges* in the *context*.
- If the item is `AllClients`, the effect is as if the actions described for `FutureClients` are performed, followed by the actions for `CurrentClients`.

The `"Alloc"` error results when the server is unable to allocate the necessary resources.

`RecordRegisterClients`

context: RC

element-header: ELEMENT_HEADER

client-specifiers: LISTofCLIENTSPEC

ranges: LISTofRECORDRANGE

Errors: `Match`, `Value`, `RecordContext`, `Alloc`

This request registers the set of clients in *client-specifiers* with the given *context* and specifies the protocol to intercept for those clients. The header preceding each recorded protocol element is set as specified by *element-header*. These flags affect the entire context; their effect is not limited to the clients registered by this request. Typically, this request is used by a recording client over the control connection.

If *context* does not name a valid RC, a `"RecordContext"` error results. If any of the values in *element-header* or *ranges* is invalid, a `"Value"` error results. Duplicate items in the list of *client-specifiers* are ignored. If any item in the list of *client-specifiers* is not a valid CLIENTSPEC, a `"Match"` error results. If the *context* is enabled and the XID of the enabling connection is specified, a `"Match"` error results. Otherwise, each item in the *client-specifiers* list is processed as follows:

- If the item is an XIDBASE identifying a particular client, the specified client is registered with the *context* if it is not already registered. The protocol to intercept for the client is then set to *ranges*.
- If the item is `CurrentClients`, all existing clients that are not already registered with the specified *context*, except the enabling connection if the *context* is enabled, are registered at this time. The protocol to intercept for all clients registered with the *context* is then set to *ranges*.
- If the item is `FutureClients`, all clients that connect to the server after this request executes will be automatically registered with the *context*. The protocol to intercept for such clients will be set to *ranges* in the *context*. The set of clients that are registered with the *context* and their corresponding sets of protocol to intercept are left intact.
- If the item is `AllClients`, the effect is as if the actions described for `FutureClients` are performed, followed by the actions for `CurrentClients`.

The `"Alloc"` error results when the server is unable to allocate the necessary resources.

`RecordUnregisterClients`

context: RC

client-specifiers: LISTofCLIENTSPEC

Errors: `Match` , `RecordContext`

This request removes the set of clients in *client-specifiers* from the given *context*'s set of registered clients. Typically, this request is used by a recording client over the control connection.

If *context* does not name a valid RC, a "`RecordContext`" error results. Duplicate items in the list of *client-specifiers* are ignored. If any item in the list is not a valid CLIENTSPEC, a "`Match`" error results. Otherwise, each item in the *client-specifiers* list is processed as follows:

- If the item is an XIDBASE identifying a particular client, and the specified client is currently registered with the *context*, it is unregistered, and the set of protocol to intercept for the client is deleted from the *context*. If the specified client is not registered with the *context*, the item has no effect.
- If the item is `CurrentClients` , all clients currently registered with the *context* are unregistered from it, and their corresponding sets of protocol to intercept are deleted from the *context*.
- If the item is `FutureClients` , clients that connect to the server after this request executes will not automatically be registered with the *context*. The set of clients that are registered with this context and their corresponding sets of protocol that will be intercepted are left intact.
- If the item is `AllClients` , the effect is as if the actions described for `FutureClients` are performed, followed by the actions for `CurrentClients` .

A client is unregistered automatically when it disconnects.

`RecordGetContext`

context: RC

->

enabled: BOOL

element-header: ELEMENT_HEADER

intercepted-clients: LISTofCLIENT_INFO

Errors:

`RecordContext`

This request queries the current state of the specified *context* and is typically used by a recording client over the control connection. The *enabled* field specifies the state of data transfer between the extension and the recording client, and is either enabled (`True`) or disabled (`False`). The initial state is disabled. When enabled, all core X protocol and extension protocol received from (requests) or sent to (replies, errors, events) a particular client, and requested to be intercepted by the recording client, is reported to the recording client over the data connection. The *element-header* specifies the header that precedes each recorded protocol element. The *intercepted-clients* field specifies the list of clients currently being recorded and the protocol associated with each client. If future clients will be automatically registered with the context, one of the returned CLIENT_INFO structures has a *client-resource* value of FutureClients and an *intercepted-protocol* giving the protocol to intercept for future clients. Protocol ranges may be decomposed, coalesced, or otherwise modified by the server from how they were specified by the client. All CLIENTSPECs registered with the server are returned, even if the RECORDRANGE(s) associated with them specify no protocol to record.

When the *context* argument is not valid, a `RecordContext` error results.

`RecordEnableContext`

context: RC

->+

category: {`FromServer`, `FromClient`, `ClientStarted`, `ClientDied`, `StartOfData`, `EndOfData`}

element-header: ELEMENT_HEADER

client-swapped: BOOL

id-base: XIDBASE

server-time: TIMESTAMP

recorded-sequence-number: CARD32

data: LISTofBYTE

Errors: `Match`, `RecordContext`

This request enables data transfer between the recording client and the extension and returns the protocol data the recording client has previously expressed interest in. Typically, this request is executed by the recording client over the data connection.

If the client is registered on the *context*, it is unregistered before any recording begins.

Once the server receives this request, it begins intercepting and reporting to the recording client all core and extension protocol received from or sent to clients registered with the RC that the recording client has expressed interest in. All intercepted protocol data is returned in the byte-order of the recorded client. Therefore, recording clients are responsible for all byte swapping, if required. More than one recording client cannot enable data transfer on the same RC at the same time. Multiple intercepted requests, replies, events and errors might be packaged into a single reply before being returned to the recording clients.

The *category* field determines the possible types of the data. When a context is enabled, the server will immediately send a reply of category `StartOfData` to notify the client that recording is enabled. A category of `FromClient` means the data are from the client (requests); `FromServer` means data are from the server (replies, errors, events, or device events). For a new client, the category is `ClientStarted` and the data are the connection setup reply. When the recorded client connection is closed, *category* is set to the value `ClientDied` and no protocol is included in this reply. When the disable request is made over the control connection, a final reply is sent over the data connection with category `EndOfData` and no protocol.

The *element-header* field returns the value currently set for the context, which tells what header information precedes each recorded protocol element in this reply.

The *client-swapped* field is `True` if the byte order of the protocol being recorded is swapped relative to the recording client; otherwise, *client-swapped* is `False` . The recorded protocol is in the byte order of the client being recorded; device events are in the byte order of the recording client. For replies of category `StartOfData` and `EndOfData` the *client-swapped* bit is set according to the byte order of the server relative to the recording client. The *id-base* field is the resource identifier base sent to the client from the server in the connection setup reply, and hence, identifies the client being recorded. The *id-base* field is 0 (zero) when the protocol data being returned are device events. The *server-time* field is set to the time of the server when the first protocol element in this reply was intercepted. The *server-time* of reply N+1 is greater than or equal to the *server-time* of reply N, and is greater than or equal to the time of the last protocol element in reply N.

The *recorded-sequence-number* field is set to the sequence number of the recorded client's most recent request processed by the server.

The *data* field contains the raw protocol data being returned to the recording client. If requested by the *element-header* of this record context, each protocol element may be preceded by a 32-bit timestamp and/or a 32-bit sequence number. If present, both the timestamp and sequence number are always in the byte order of the recording client.

For the core X events `KeyPress` , `KeyRelease` , `ButtonPress` , and `ButtonRelease` , the fields of a device event that contain valid information are *time* and *detail*. For the core X event `MotionNotify` , the fields of a device event that contain valid information are *time*, *root*, *root-x* and *root-y*. The *time* field refers to the time the event was generated by the device.

For the extension input device events `DeviceKeyPress` , `DeviceKeyRelease` , `DeviceButtonPress` , and `DeviceButtonRelease` , the fields of a device event that contain valid information are *device*, *time* and *detail*. For `DeviceMotionNotify` , the valid device event fields are *device* and *time*. For the extension input device events `ProximityIn` and `ProximityOut` , the fields of a device event that contain valid information are *device* and *time*. For the extension input device event `DeviceValuator` , the fields of a device event that contain valid information are *device*, *num_valuators*, *first_valuator*, and *valuators*. The *time* field refers to the time the event was generated by the device.

The error `"Match"` is returned when data transfer is already enabled. When the *context* argument is not valid, a `RecordContext` error results.

`RecordDisableContext`

context: RC

Errors: `RecordContext`

This request is typically executed by the recording client over the control connection. This request directs the extension to immediately send any complete protocol elements currently buffered, to send a final reply with category `EndOfData` , and to discontinue data transfer between the extension and the recording client. Protocol reporting is disabled on the data connection that is currently enabled for the given *context*. Once the extension completes processing this request, no additional recorded protocol will be reported to the recording client. If a data connection is not currently enabled when this request is executed, then this request has no affect on the state of data transfer. An RC is disabled automatically when the connection to the enabling client is closed down.

When the *context* argument is not valid, a `RecordContext` error results.

`RecordFreeContext`

- *context* RC
- Errors: `RecordContext`

This request deletes the association between the resource ID and the RC and destroys the RC. If a client has enabled data transfer on this *context*, the actions described in `RecordDisableContext` are performed before the *context* is freed.

An RC is destroyed automatically when the connection to the creating client is closed down and the close-down mode is `DestroyAll`. When the *context* argument is not valid, a `RecordContext` error results.

Chapter 4. Encoding

Please refer to the X11 Protocol Encoding document as this document uses conventions established there.

The name of this extension is "RECORD".

Types

RC: CARD32

```
RANGE8
     1     CARD8           first
     1     CARD8           last
```

```
RANGE16
     2     CARD16           first
     2     CARD16           last
```

```
EXTRANGE
     2     RANGE8           major
     4     RANGE16          minor
```

```
RECORDRANGE
     2     RANGE8           core-requests
     2     RANGE8           core-replies
     6     EXTRANGE         ext-requests
     6     EXTRANGE         ext-replies
     2     RANGE8           delivered-events
     2     RANGE8           device-events
     2     RANGE8           errors
     1     BOOL             client-started
     1     BOOL             client-died
```

```
ELEMENT_HEADER
     1     CARD8
          0x01     from-server-time
          0x02     from-client-time
          0x04     from-client-sequence
```

XIDBASE: CARD32

```
CLIENTSPEC
     4    XIDBASE  client-id-base
          1        CurrentClients
          2        FutureClients
          3        AllClients
```

```
CLIENT_INFO
     4    CLIENTSPEC          client-resource
     4    CARD32              n, number of record ranges in
                                 intercepted-protocol
     24n  LISTofRECORDRANGE   intercepted-protocol
```

Errors

```
RecordContext
     1     0                  Error
     1     CARD8              extension's base error code + 0
     2     CARD16             sequence number
     4     CARD32             invalid record context
     24                       unused
```

Requests

```
RecordQueryVersion
     1     CARD8      major opcode
     1     0          minor opcode
     2     2          request length
     2     CARD16     major version
     2     CARD16     minor version
 =>
     1     1          Reply
     1                unused
     2     CARD16     sequence number
     4     0          reply length
     2     CARD16     major version
     2     CARD16     minor version
     20               unused
```

```
RecordCreateContext
     1     CARD8                 major opcode
     1     1                     minor opcode
     2     5+m+6n                request length
     4     RC                    context
     1     ELEMENT_HEADER        element-header
     3                           unused
     4     CARD32                m, number of client-specifiers
     4     CARD32                n, number of ranges
     4m    LISTofCLIENTSPEC      client-specifiers
     24n   LISTofRECORDRANGE     ranges
```

```
RecordRegisterClients
     1     CARD8                 major opcode
     1     2                     minor opcode
     2     5+m+6n                request length
```

```
     4     RC                    context
     1     ELEMENT_HEADER        element-header
     3                           unused
     4     CARD32                m, number of client-specifiers
     4     CARD32                n, number of ranges
     4m    LISTofCLIENTSPEC      client-specifiers
     24n   LISTofRECORDRANGE     ranges


RecordUnregisterClients
     1     CARD8                 major opcode
     1     3                     minor opcode
     2     3+m                   request length
     4     RC                    context
     4     CARD32                m, number of client-specifiers
     4m    LISTofCLIENTSPEC      client-specifiers


RecordGetContext
     1     CARD8                 major opcode
     1     4                     minor opcode
     2     2                     request length
     4     RC                    context
 =>
     1     1                     Reply
     1     BOOL                  enabled
     2     CARD16                sequence number
     4     j                     reply length
     1     ELEMENT_HEADER        element-header
     3                           unused
     4     CARD32                n, number of intercepted-clients
     16                          unused
     4j    LISTofCLIENT_INFO     intercepted-clients


RecordEnableContext
     1     CARD8                 major opcode
     1     5                     minor opcode
     2     2                     request length
     4     RC                    context
 =>+
     1     1                     Reply
     1                           category
           0     FromServer
           1     FromClient
           2     ClientStarted
           3     ClientDied
           4     StartOfData
           5     EndOfData
     2     CARD16                sequence number
     4     n                     reply length
     1     ELEMENT_HEADER        element-header
     1     BOOL                  client-swapped
     2                           unused
```

```
     4     XIDBASE               id-base
     4     TIMESTAMP             server-time
     4     CARD32                recorded-sequence-number
     8                           unused
     4n    BYTE                  data


RecordDisableContext
     1     CARD8                 major opcode
     1     6                     minor opcode
     2     2                     request length
     4     RC                    context


RecordFreeContext
     1     CARD8                 major opcode
     1     7                     minor opcode
     2     2                     request length
     4     RC                    context
```