X Record Extension Library

X Consortium Standard

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

X Record Extension Library: X Consortium Standard

by Martha Zimet and Stephen Gildea

X Version 11, Release 7.7

Version 1.13

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 © 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 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.

Table of Contents

Chapter 1. Record Extension Overview

The purpose of this extension is to support the recording and reporting of all core X protocol and arbitrary X extension protocol. This first section gives an overview of the Record extension. The following sections describe how to use the Record extension library.

Synchronous Playback

Environment information is generally provided to an X-based playback mechanism, which might use the XTest extension to synthesize input events. This synchronization information defines the X state prior to event synthesis (for example, location of the cursor, window locations and sizes, installed colormap, window manager running, and so on) and the consequences that occur after the playback mechanism synthesizes the event. If the user moves the mouse into the icon window and presses and releases a mouse button, the device events `MotionNotify`, `ButtonPress`, and `ButtonRelease` are generated by the X server. Because X follows an event-driven model, there are consequences that follow from the user actions, or device events, that are in the form of X protocol. As a result of the previous user actions, the client could generate requests such as `ImageText8` and `PolyLine` to the X server, or the X server could send non-device events such as `Expose` and `MapNotify` to the client window. Both the requests and non-device events that result from user actions are known as consequences, which can be used as a synchronization, or control point, during playback. That is, the playback mechanism does not generate a specific synthesized event until its matching synchronization condition occurs (for example, the window is mapped or unmapped, the cursor changes, a text string displays, and so on)

Because it cannot be predicted what synchronization information is required during playback, the Record extension makes no assumptions about the intended use of the recorded data. Facilities exist to record any core X protocol or X extension protocol. Therefore, Record does not enforce a specific synchronization methodology.

Design Approach

The design approach of the extension is to record core X protocol and arbitrary X extension protocol entirely within the X server itself. When the extension has been requested to record specific protocol by one or more recording clients, the protocol data is formatted and returned to the recording clients. The extension provides a mechanism for capturing all events, including input device events that do not go to any clients.

Record Clients

The recommended communication model for a Record application is to open two connections to the server—one connection for recording control and one connection for reading recorded protocol data.

Information about recording (for example, what clients to record, what protocol to record for each client, and so on) is stored in resources called record contexts (type `XRecordContext`). Most Record extension functions take a record context as an argument. Although in theory it is possible to share record contexts between applications, it is expected that applications will use their own context when performing recording operations.

A client that wishes to record X protocol does so through the library functions defined in *Library Extension Requests* A typical sequence of requests that a client would make is as follows:

- `XRecordQueryVersion`

  query the extension protocol version.

- `XRecordCreateContext`

  request that the server create a record context for access by this client, and express interest in clients and protocol to be recorded. This request returns an `XRecord-Context`, which is an XID that is used by most other extension requests to identify the specified context.

- `XRecordEnableContext`

  begin the recording and reporting of protocol data.

- `XRecordDisableContext`

  end the recording and reporting of protocol data.

- `XRecordFreeContext`

  free the record context.

The header for this library is <`X11/extensions/record.h`>. All identifiers defined in the interface are supplied by this header and are prefixed with "XRecord". The `Xtst` library contains the `XRecord` functions.

Chapter 2. Common Arguments

The Record extension functions `XRecordCreateContext` and `XRecordRegisterClients` allow applications to specify the following:

- Individual clients or sets of clients to record
- Ranges of core X protocol and X extension protocol to record for each client

Protocol in the ranges specified by the recording client will be recorded by the server. The device_events protocol type can be specified by a recording client although it may not be sent to a recorded client. The device_events type differs from delivered_events, which also can be specified by a recording client; delivered_events are actually delivered to one or more clients. These event types are discussed in Protocol Ranges

The Record extension functions `XRecordCreateContext` and `XRecordRegisterClients` have the common arguments datum_flags, clients, and ranges, which specify whether server time and/or client sequence number should precede protocol elements, the clients or client set to record, and the protocol ranges to record, respectively. These are discussed in the following sections.

Datum Flags

The datum_flags argument is a set of flags OR'ed together to specify options for the record context. Specify zero to disable all the options.

The `XRecordFromServerTime` flag specifies that `XRecordInterceptData` structures with a category of `XRecordFromServer` will have a server_time field specific to each protocol element.

The `XRecordFromClientTime` flag specifies that `XRecordInterceptData` structures with a category of `XRecordFromClient` will have a server_time field specific to each protocol element.

The `XRecordFromClientSequence` flag specifies that `XRecordInterceptData` structures with a category of `XRecordFromClient` or `XRecordClientDied` will have a valid client_seq field.

Selecting Clients

The clients argument is a pointer to an array of `XRecordClientSpec`. `XRecordClientSpec` is an integral type that holds a resource ID, a client resource ID base, or one of the client set constants defined below.

Duplicate elements in the array are ignored by the functions, and if any element in the array is not valid, a `BadMatch` error results. A resource ID references the client that created that resource. The client set may be one of the following constants: `XRecordCurrentClients`, `XRecordFutureClients`, or `XRecordAllClients`.

If the element in the array identifies a particular client, protocol specified by the ranges argument will be recorded by the server. The recorded protocol data will not be returned to the recording client until the record context has been enabled. This is described in Data Transfer

If the element is `XRecordCurrentClients`, the protocol ranges specified by the ranges argument, except for device_events, are associated with each current client connection. If the element is `XRecordFutureClients`, the protocol ranges specified by the ranges argument are associated with each new client connection. If the element is `XRecordAllClients`, the protocol ranges specified by the ranges argument are associated with each current client connection and with each new client connection. When the context is enabled, the data connection is unregistered if it was registered. If the context is enabled, `XRecordCurrentClients` and `XRecordAllClients` silently exclude the recording data connection. It is an error to explicitly register the data connection.

Protocol Ranges

The functions `XRecordCreateContext` and `XRecordRegisterClients` have another common argument, ranges, which is an array of pointers to `XRecordRange` structures. Each structure contains ranges of numeric values for each of the protocol types that can be specified and recorded individually by the Record extension. An `XRecordRange` structure must be allocated by the Record library using the `XRecordAllocRange` function.

The `XRecordRange` typedef is a structure with the following members:

```
XRecordRange:
    XRecordRange8 core_requests      /* core X requests */
    XRecordRange8 core_replies       /* core X replies */
    XRecordExtRange ext_requests     /* extension requests */
    XRecordExtRange ext_replies      /* extension replies */
    XRecordRange8 delivered_events   /* delivered core and ext events */
    XRecordRange8 device_events      /* all core and ext device events */
    XRecordRange8 errors             /* core X and X ext errors */
    Bool client_started              /* connection setup reply from server */
    Bool client_died                 /* notification of client disconnect */
```

The types used in `XRecordRange` members are defined as follows. The `XRecordRange8` typedef is a structure with the following members:

```
XRecordRange8:
    unsigned char first
    unsigned char last
```

The `XRecordRange16` typedef is a structure with the following members:

```
XRecordRange16:
    unsigned short first
    unsigned short last
```

The `XRecordExtRange` typedef is a structure with the following members:

```
XRecordExtRange:
    XRecordRange8  ext_major
```

XRecordRange16 ext_minor

If any of the values specified in `XRecordRange` is invalid, a `BadValue` error results.

The core_requests member specifies the range of core X protocol requests to record. Core X protocol requests with a major opcode that is between first and last, inclusive, will be recorded. A `BadValue` error results if the value of first is greater than the value of last. If the values of both first and last are zero, no core X protocol requests will be recorded.

The core_replies member specifies the range of replies resulting from core X protocol requests to record. Replies that result from core X protocol requests with a major opcode between first and last, inclusive, will be recorded. A `BadValue` error results if the value of first is greater than the value of last. If the values of both first and last are zero, no core X protocol replies will be recorded.

The ext_requests member specifies the range of X extension requests to record. X extension requests with a major opcode between ext_major.first and ext_major.last, and with a minor opcode between ext_minor.first and ext_minor.last, inclusive, will be recorded. A `BadValue` error results if the value of ext_major.first is greater than the value of ext_major.last or if the value of ext_minor.first is greater than the value of ext_minor.last. If the values of both ext_major.first and ext_major.last are zero, no X extension requests will be recorded.

The ext_replies member specifies the range of replies resulting from X extension requests to record. Replies that result from an X extension request with a major opcode between ext_major.first and ext_major.last, and a minor opcode that is between ext_minor.first and ext_minor.last will be recorded. A `BadValue` error results if the value of ext_major.first is greater than the value of ext_major.last or if the value of ext_minor.first is greater than the value of ext_minor.last. If the values of both ext_major.first and ext_major.last are zero, no X extension replies will be recorded.

The delivered_events member specifies the range of both core X events and X extension events to record. These events are delivered to at least one client. Core X events and X extension events with a code value between first and last inclusive will be recorded. A `BadValue` error results if the value of first is greater than the value of last. If the values of first and last are zero, no events will be recorded.

The device_events member specifies the range of both core X device events and X extension device events to record. These events may or may not be delivered to a client. Core X device events and X extension device events with a code value between first and last inclusive that are not delivered to any clients will be recorded. A `BadValue` error results if the value of first is greater than the value of last. A `BadValue` error results if first is less than two or last is less than two, except that if first and last are zero, no events will be recorded.

The errors member specifies the range of both core X errors and X extension errors to record. Core X errors and X extension errors with a code value between first and last inclusive will be recorded. A `BadValue` error results if the value of first is greater than the value of last. If the values of first and last are zero, no errors will be recorded.

A value of `True` for the client_started member specifies the connection setup reply from the server to new clients. If `False` the connection setup reply is not specified by this `XRecordRange`

A value of `True` for the client_died member specifies notification when a client disconnects. If `False` notification when a client disconnects is not specified by this `XRecordRange`

Chapter 3. Library Extension Requests

Recording operations are accessed by programs through the use of new protocol requests. The following functions are provided as extensions to Xlib. An Xlib error results if an extension request is made to an X server that does not support the Record extension. Note that any of the extension protocol requests may generate `BadAlloc` or `BadLength` errors.

Query Extension Version

An application uses the `XRecordQueryVersion` function to determine the version of the Record extension protocol supported by an X server.

```
Status XRecordQueryVersion( *display,  cmajor_return,  cminor_return);
```

display	Returns the connection to the X server.
cmajor_return	Returns the extension protocol major version in use.
cminor_return	Returns the extension protocol minor version in use.

The `XRecordQueryVersion` function returns the major and minor protocol version numbers supported by the server. `XRecordQueryVersion` returns nonzero (success) only if the returned version numbers are common to both the library and the server; otherwise, it returns zero.

Create and Modify Context

An application uses the `XRecordCreateContext` function to create a record context. At the time the record context is created by the recording client, the clients to be recorded and the protocol to record for each client may be specified.

```
XRecordContext XRecordCreateContext( *display,  datum_flags,  *clients,
nclients,  *ranges,  nranges);
```

display	Returns the connection to the X server.
datum_flags	Specifies whether detailed time or sequence info should be sent.
clients	Specifies the clients to record.
nclients	Specifies the number of clients.
ranges	Specifies the protocol ranges to record.
nranges	Specifies the number of protocol ranges.

The `XRecordCreateContext` function creates a record context and returns an `XRecordContext` which is then used in the other Record library calls. This request is typically executed by the recording client over its control connection to the X server. The datum_flags specifies whether server time and/or client sequence number should precede protocol elements recorded by context (Datum Flags). When a clients element identifies a particular client, the client is added to the context and the protocol to record for that client is set to the union of all ranges. When a clients element is `XRecordCurrentClients` `XRecordFutureClients` or `XRecordAllClients` the actions described in Selecting Clients are performed.

`XRecordCreateContext` returns zero if the request failed. `XRecordCreateContext` can generate `BadIDChoice` `BadMatch` and `BadValue` errors.

The ranges argument is an `XRecordRange` array, that is, an array of pointers. The structures the elements point to shall be allocated by calling `XRecordAllocRange`

XRecordRange *
XRecordAllocRange(void)

The `XRecordAllocRange` function allocates and returns an `XRecordRange` structure. The structure is initialized to specify no protocol. The function returns NULL if the structure allocation fails. The application can free the structure by calling `XFree`

Additions

An application uses the `XRecordRegisterClients` function to modify a previously created record context, by adding clients or modifying the recorded protocol, typically over its control connection to the X server.

`Status` **`XRecordRegisterClients`**`(` *`*display`*`,` *`context`*`,` *`datum_flags`*`,` *`*clients`*`,` *`nclients`*`,` *`*ranges`*`,` *`nranges`*`);`

display	Returns the connection to the X server.
context	Specifies the record context to modify.
datum_flags	Specifies whether detailed time or sequence info should be sent.
clients	Specifies the clients to record.
nclients	Specifies the number of clients.
ranges	Specifies the protocol ranges to record.
nranges	Specifies the number of protocol ranges.

The datum_flags specifies whether server time and/or client sequence number should precede protocol elements for all clients recorded by context (See Datum Flags). When a clients element identifies a particular client and the client is not yet targeted for recording in the given context, the client is added to the set of clients to record, and the protocol to record for that client is set to the union of all ranges. When the client is already targeted for recording, the protocol to record for that client is set to the union of all ranges. When the element is `XRecordCurrentClients` `XRecordFutureClients` or `XRecordAllClients` the actions described in Selecting Clients are performed.

`XRecordRegisterClients` returns zero if the request failed; otherwise, it returns nonzero.

`XRecordRegisterClients` can generate `XRecordBadContext` `BadMatch` and `BadValue` errors.

Deletions

An application uses the `XRecordUnregisterClients` function to delete clients from a previously created record context, typically over its control connection to the X server.

`Status` **`XRecordUnRegisterClients`**`(` *`*display`*`,` *`context`*`,` *`*clients`*`,` *`nclients`*`);`

display	Returns the connection to the X server.
context	Specifies the record context to modify.
clients	Specifies the clients to stop recording.
nclients	Specifies the number of clients.

When an element in clients identifies a particular client, and the specified client is already targeted for recording in the given context, the client and the set of protocol to record for that client are deleted from the context. If the specified client is not targeted for recording, then no action is performed.

When the element is `XRecordCurrentClients` all clients currently targeted for recording in context and their corresponding sets of protocol to record are deleted from context.

When the item is `XRecordFutureClients` any future client connections will not automatically be targeted for recording in context.

When the element is `XRecordAllClients` all clients currently targeted for recording in context and their corresponding sets of protocol to record are deleted from context. Any future client connections will not automatically be targeted for recording in context.

`XRecordUnregisterClients` returns zero if the request failed; otherwise, it returns nonzero.

`XRecordUnregisterClients` can generate `XRecordBadContext` `BadMatch` and `BadValue` errors.

Query Context State

An application uses the `XRecordGetContext` function to query the current state of a record context, typically over its control connection to the X server.

`Status` **`XRecordGetContext`**`(` *`*display`*`,` *`context`*`,` *`**state_return`*`);`

display	Specifies the connection to the X server.
context	Specifies the record context to query.
state_return	Specifies the address of a variable into which the function stores a pointer to the current state of the record context.

The `XRecordState` typedef returned by `XRecordGetContext` is a structure with the following members:

XRecordState:
    Bool   enabled
    int   datum_flags
    unsigned long   nclients
    XRecordClientInfo **client_info

The enabled member is set to the state of data transfer and is `True` when the recording client has asked that recorded data be sent; otherwise it is `False` The datum_flags member is set to the value of these flags for this context. The nclients member is set to the number of `XRecordClientInfo` structures returned. The client_info member is an array of pointers to `XRecordClientInfo` structures that contain the protocol to record for each targeted client. The `XRecordClientInfo` typedef is a structure with the following members:

XRecordClientInfo:
    XRecordClientSpec client
    unsigned long nranges
    XRecordRange **ranges

The client member either identifies a client targeted for recording or is set to `XRecordFutureClients` to describe how future clients will be automatically targeted for recording. The nranges member is set to the number of protocol ranges to be recorded for the specified client. The ranges member is an array of pointers to `XRecordRange` structures, which specify the protocol ranges to record.

`XRecordGetContext` returns zero if the request failed; otherwise, it returns nonzero. The context argument must specify a valid `XRecordContext` or a `XRecordBadContext` error results.

Recording clients should use the `XRecordFreeState` function to free the state data returned by `XRecordGetContext`

`void` **`XRecordFreeState`**`(` *`*state`*`);`

state   Specifies the structure that is to be freed.

`XRecordFreeState` frees the data pointed to by state. If the argument does not match an `XRecordState` pointer returned from a successful call to `XRecordGetContext` or if `XRecordFreeState` has already been called with it, the behavior is undefined.

Data Transfer

An application uses the `XRecordEnableContext` and `XRecordDisableContext` functions to change the state of data transfer between the X server and the recording client. These functions allow the application to start recording and reporting of protocol data and to stop recording and reporting of protocol data, respectively.

Enable Context

To direct the X server to record and report protocol, a program uses `XRecordEnableContext` typically over its data connection to the X server. The reporting of recorded protocol back to the recording client is handled by the following data structures and procedure definitions. Each recorded protocol element is reported to the recording client through an `XRecordInterceptData` typedef, a structure with the following members:

XRecordInterceptData:

XID	id_base
Time	server_time
unsigned long	client_seq
int	category
Bool	client_swapped
unsigned char	*data
unsigned long	data_len

The id_base member is set to the resource identifier base sent to the client in the connection setup reply and therefore identifies the client being recorded, except when the recorded protocol data is a device event that may have not been delivered to a client. In this case, id_base is set to zero. The server_time member is set to the time of the server when the protocol was recorded. It is the time that was attached to this protocol element in the reply, if so specified by datum_flags, or else the time from the header of the reply that contained this protocol element. The client_seq member is the sequence number of the recorded client's most recent request processed by the server at the time this protocol element was recorded, if this information were included in the recorded data; otherwise client_seq is 0. The category member is set to one of the following values: `XRecordStartOfData` `XRecordFromServer` `XRecordFromClient` `XRecordClientStarted` `XRecordClientDied` or `XRecordEndOfData` `XRecordStartOfData` is immediately sent as the first reply to confirm that the context is enabled. `XRecordFromClient` indicates the protocol data is from the recorded client to the server (requests). `XRecordFromServer` indicates the protocol data is from the server to the recorded client (replies, errors, events, or device events). `XRecordClientStarted` indicates that the protocol data is the connection setup reply from the server. `XRecordClientDied` indicates that the recorded client has closed its connection to the X server; there is no protocol data. `XRecordEndOfData` indicates that the context has been disabled and that this is the last datum. It does not correspond to any protocol or state change in a recorded client. There is no protocol data.

The client_swapped member is set to `True` if the byte order of the client being recorded is swapped relative to the recording client; otherwise, it is set to `False` All recorded protocol data is returned in the byte order of the recorded client. Therefore, recording clients are responsible for all byte swapping, if required. Device events are in the byte order of the recording client. For replies of category `XRecordStartOfData` and `XRecordEndOfData` client_swapped is set according to the byte order of the server relative to the recording client.

The data member contains the actual recorded protocol data. When category is set to `XRecordStartOfData` `XRecordClientDied` or `XRecordEndOfData` no protocol data are contained in data.

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 data_len member is set to the length of the actual recorded protocol data in 4-byte units.

When the context has been enabled, protocol data the recording client has previously expressed interest in is recorded and returned to the recording client via multiple replies. Because the X server batches the recorded data, more than one protocol element may be contained in the same reply packet. When a reply is received, a procedure of type `XRecordInterceptProc` is called for each protocol element in the reply.

```
typedef void (*XRecordInterceptProc)( closure,  *recorded_data);
```

closure	Pointer that was passed in when the context was enabled.
recorded_data	A protocol element recorded by the server extension.

This callback may use the control display connection (or any display connection other than the data connection).

Recording clients should use the `XRecordFreeData` function to free the `XRecordInterceptData` structure.

```
Status XRecordEnableContext( *display,  context,  callback,  closure);
```

display	Specifies the connection to the X server.
context	Specifies the record context to enable.
callback	Specifies the function to be called for each protocol element received.
closure	Specifies data passed to callback.

`XRecordEnableContext` enables data transfer between the recording client and the X server. All core and extension protocol received from or sent to targeted clients that the recording client has expressed interest in will be recorded and reported to the recording client.

`XRecordEnableContext` returns zero if the request failed; otherwise, it returns nonzero. The context argument must specify a valid `XRecordContext` or a `XRecordBadContext` error results. The error `BadMatch` results when data transfer is already enabled on the given context.

Enable Context Asynchronously

Because `XRecordEnableContext` does not return until `XRecordDisableContext` is executed on the control connection, a nonblocking interface in addition to `XRecor-`

`dEnableContext` is provided. This interface also enables data transfer; however, it does not block.

This interface is defined as follows:

`Status` **`XRecordEnableContextAsync`**`(` *`*display`*`,` *`context`*`,` *`callback`*`,` *`closure`*`);`

display Specifies the connection to the X server.

context Specifies the record context to enable.

callback Specifies the function to be called for each protocol element received.

closure Specifies data passed to callback.

`XRecordEnableContextAsync` enables data transfer between the recording client and the X server just as `XRecordEnableContext` does. Unlike `XRecordEnableContext` it does not wait for the context to be disabled before returning; `XRecordEnableContextAsync` returns as soon as the `XRecordStartOfData` reply has been received and processed.

`XRecordEnableContextAsync` returns zero if it could not allocate the necessary memory and nonzero if it sent the request successfully to the server. The context argument must specify a valid `XRecordContext` or a `XRecordBadContext` error results. The error `BadMatch` results when data transfer is already enabled.

Each time it reads data from the server connection, Xlib will check for incoming replies and call *callback* as necessary. The application may direct Xlib explicitly to check for Record data with the `XRecordProcessReplies` function.

`void` **`XRecordProcessReplies`**`(` *`*display`*`);`

display Specifies the connection to the X server.

`XRecordProcessReplies` will check for any replies that have not yet been processed by the application. The asynchronous callback will be called as appropriate. `XRecordProcessReplies` returns when all immediately available replies have been processed. It does not block.

To free the data passed to the `XRecordInterceptProc` callback, use `XRecordFreeData`

`void` **`XRecordFreeData`**`(` *`*data`*`);`

data Specifies the structure that is to be freed.

`XRecordFreeData` frees the data pointed to by data. If the argument does not match an `XRecordInterceptData` pointer earlier passed to an `XRecordInterceptProc` callback or if `XRecordFreeData` has already been called with it, the behavior is undefined.

Disable Context

To direct the X server to halt the reporting of recorded protocol, the program executes `XRecordDisableContext` typically over its control connection to the X server.

`Status` **`XRecordDisableContext`**`(` *`*display,`* *`context`*`);`

display Specifies the connection to the X server.

context Specifies the record context to disable.

The `XRecordDisableContext` function disables context, stopping all recording over its data connection. Any complete protocol elements for context that were buffered in the server will be sent to the recording client rather than being discarded. If a program attempts to disable an `XRecordContext` that has not been enabled, no action will take place.

`XRecordDisableContext` returns zero if the request failed; otherwise, it returns nonzero. The context argument must specify a valid `XRecordContext` or an `XRecordBadContext` error results.

ID Base Mask

To determine the mask the server uses for the client ID base, use `XRecordIdBaseMask`

`XID` **`XRecordIdBaseMask`**`(` *`*display`*`);`

display Specifies the connection to the X server.

The `XRecordIdBaseMask` function returns the resource ID mask passed to the client by the server at connection setup.

Free Context

Before terminating, the program should request that the server free the record context. This is done with the `XRecordFreeContext` function, typically over the record client's control connection to the X server.

`Status` **`XRecordFreeContext`**`(` *`*display,`* *`context`*`);`

display Specifies the connection to the X server.

context Specifies the record context to free.

The `XRecordFreeContext` function frees the given context for the requesting client. Freeing a record context releases the clients targeted for recording and their respective protocol ranges to record. If protocol data is being reported to the recording client, generally over the data connection to the X server, the reporting ceases as if `XRecordDisableContext` had been called on the given context. When a program terminates without freeing its record context, the X server will automatically free that context on behalf of the client.

`XRecordFreeContext` returns zero if the request failed; otherwise,it returns nonzero. The context argument must specify a valid `XRecordContext` or a `XRecordBadContext` error results.