X SYNCHRONIZATION EXTENSION

Version 3.0

X Consortium Standard

X Version 11, Release 6

Tim Glauert
thg@cam-orl.co.uk
Olivetti Research / MultiWorks

Dave Carver
dcc@athena.mit.edu
Digital Equipment Corporation,
MIT / Project Athena

Jim Gettys
jg@crl.dec.com
Digital Equipment Corporation,
Cambridge Research Laboratory

David P. Wiggins
dpw@x.org
X Consortium, Inc.

Copyright 1991 by Olivetti Research Limited, Cambridge, England and Digital Equipment Corporation, Maynard, Massachusetts.

Permission to use, copy, modify, and distribute this documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies. Olivetti, Digital, MIT, and the X Consortium make 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 (c) 1991 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.

Synchronization

The core X protocol makes no guarantees about the relative order of execution of requests for different clients. This means that any synchronization between clients must be done at the client level in an operating-system dependent and network dependent manner. Even if there was an accepted standard for such synchronization, the use of a network introduces unpredictable delays between the synchronization of the clients and the delivery of the resulting requests to the X server.

The core X protocol also makes no guarantees about the time at which requests are executed, which means that all clients with real-time constraints must implement their timing on the host computer. Any such timings are subject to error introduced by delays within the operating system and network, and are inefficient due to the need for round-trip requests which keep the client and server synchronized.

The synchronization extension provides primitives which allow synchronization between clients to take place entirely within the X server. This removes any error introduced by the network and makes it possible to synchronize clients on different hosts running different operating systems. This is important for multi-media applications where audio, video and graphics data streams are being synchronized. The extension also provides internal timers within the X server to which client requests can be synchronized. This allows simple animation applications to be implemented without any round-trip requests and makes best use of buffering within the client, network and server.

Description

The mechanism used by this extension for synchronization within the X server is to block the processing of requests from a client until a specific synchronization condition occurs. When the condition occurs the client is released and processing of requests continues. Multiple clients may block on the same condition to give inter-client synchronization. Alternatively, a single client may block on a condition such as an animation frame-marker.

The extension adds **Counter** and **Alarm** to the set of resources managed by the server. A counter has a 64-bit integer value which may be increased or decreased by client requests or by the server internally. A client can block by sending an **Await** request which waits until one of a set of synchronization conditions, called TRIGGERs, becomes TRUE.

The **CreateCounter** request allows a client to create a **Counter** which can be changed by explicit **SetCounter** and **ChangeCounter** requests. These

can be used to implement synchronization between different clients.

There are some counters, called **System Counters**, which are changed by the server internally rather than by client requests. The effect of any change to a system counter is not visible until the server has finished processing the current request. In other words, system counters are apparently updated in the gaps between the execution of requests rather than during the actual execution of a request. The extension provides a system counter which advances with the server time as defined by the core protocol, and it may also provide counters which advance with the real-world time or which change each time the CRT screen is refreshed. Other extensions may provide their own extension-specific system counters.

The extension provides an **Alarm** mechanism which allows clients to receive an event on a regular basis when a particular counter is changed.

Types

Please refer to the X11 Protocol specification as this document uses syntactic conventions established there and references types defined there.

The following new types are used by the extension.

INT64:	64-bit signed integer
COUNTER:	XID
VALUETYPE:	{**Absolute**,**Relative**}
TESTTYPE:	{**PositiveTransition**,**NegativeTransition**, **PositiveComparison**,**NegativeComparison**}
TRIGGER:	[*counter*:COUNTER, *value-type*:VALUETYPE, *wait-value*:INT64, *test-type*:TESTTYPE]
WAITCONDITION:	[*trigger*:TRIGGER, *event-threshold*:INT64]
SYSTEMCOUNTER:	[*name*:STRING8, *counter*:COUNTER, *resolution*:INT64]
ALARM:	XID

ALARMSTATE: {**Active**,**Inactive**,**Destroyed**}

The COUNTER type defines the client-side handle on a server **Counter**. The value of a counter is an INT64.

The TRIGGER type defines a test on a counter which is either TRUE or FALSE. The value of the test is determined by the combination of a test value, the value of the *counter* and the *test-type*.

The test value for a trigger is calculated using the *value-type* and *wait-value* fields when the trigger is initialized. If the *value-type* field is not one of the named VALUETYPE constants, the request which initialized the trigger will return a **Value** error. If the *value-type* field is **Absolute**, the test value is given by the *wait-value* field. If the *value-type* field is **Relative**, the test value is obtained by adding the *wait-value* field to the value of the counter. If the resulting test value would lie outside the range for an INT64, the request which initialized the trigger will return a **Value** error. If *counter* is **None** and the *value-type* is **Relative**, the request which initialized the trigger will return a **Match** error. If *counter* is not **None** and does not name a valid counter, a **Counter** error is generated.

If the *test-type* is **PositiveTransition**, the trigger is initialized to FALSE and it will become TRUE when the *counter* changes from a value less than the test value to a value greater than or equal to the test value. If the *test-type* is **NegativeTransition**, the trigger is initialize to FALSE and it will become TRUE when the *counter* changes from a value greater than the test value to a value less than or equal to the test value. If the *test-type* is **PositiveComparison** the trigger is TRUE if the counter is greater than or equal to the test value and FALSE otherwise. If the *test-type* is **NegativeComparison** the trigger is TRUE if the counter is less than or equal to the test value and FALSE otherwise. If the *test-type* is not one of the named TESTTYPE constants, the request which initialized the trigger will return a **Value** error. A trigger with a *counter* value of **None** and a valid *test-type* is always TRUE.

The WAITCONDITION type is simply a trigger with an associated *event-threshold*. The event threshold is used by the **Await** request to decide whether or not to generate an event to the client after the trigger has become TRUE. By setting the *event-threshold* to an appropriate value it is possible to detect the situation where an **Await** request was processed *after* the TRIGGER became TRUE, which usually indicates that the server is not processing requests as fast as the client expects.

The SYSTEMCOUNTER type provides the client with information about a **System Counter**. The *name* field is the textual name of the counter which

identifies the counter to the client. The *counter* field is the client-side handle which should be used in requests which require a counter. The *resolution* field gives the approximate step size of the system counter. This is a hint to the client that the extension may not be able to resolve two wait conditions with test values that differ by less than this step size. A microsecond clock, for example, may advance in steps of 64 microseconds so a counter based on this clock would have a *resolution* of 64.

The only system counter that is guaranteed to be present is called SERVERTIME, which counts milliseconds from some arbitrary starting point. The least-significant 32-bits of this counter track the value of Time used by the server in Events and Requests. Other system counters may be provided by different implementations of the extension. The X Consortium will maintain a registry of system counter names to avoid collisions in the name space.

An ALARM is the client-side handle on an **Alarm** resource.

Errors

Counter

This error is generated if the value for a COUNTER argument in a request does not name a defined COUNTER.

Alarm

This error is generated if the value for an ALARM argument in a request does not name a defined ALARM.

Requests

Initialize

version-major,*version-minor*: CARD8
⇒
version-major,*version-minor*: CARD8

This request must be executed before any other requests for this extension. If a client violates this rule, the results of all SYNC requests that it issues are undefined. The request takes the version number of the extension which the client wishes to use and returns the actual version number being implemented by the extension for this client. The extension may return different version numbers to a client depending of the version number supplied by that client. This request should only be executed once for each client connection.

Given two different versions of the SYNC protocol, v1 and v2, v1 is compatible with v2 if and only if $v1.version_major = v2.version_major$ and $v1.version_minor \leq v2.version_minor$. Compatible means that the functionality is fully supported in an identical fashion in the two versions.

This document describes major version 3, minor version 0 of the SYNC protocol.

ListSystemCounters

⇒
system-counters: LISTofSYSTEMCOUNTER

Errors: **Alloc**

This request returns a list of all the system counters which are available at the time the request is executed, which includes the system counters which are maintained by other extensions. The list returned by this request may change as counters are created and destroyed by other extensions.

CreateCounter

id: COUNTER
initial-value: INT64

Errors: **IDChoice**, **Alloc**

This request creates a counter and assigns the identifier *id* to it. The counter value is initialized to *initial-value* and there are no clients waiting on the counter.

DestroyCounter

counter: COUNTER

Errors: **Counter**,**Access**

This request destroys the given counter and sets the *counter* fields for all triggers which specify this counter to **None**. All clients waiting on the counter are released and a **CounterNotify** event with the *destroyed* field set to TRUE is sent to each waiting client, regardless of the *event-threshold*. All alarms specifying the counter become **Inactive** and an **AlarmNotify** event with a *state* field of **Inactive** is generated. A counter is destroyed automatically when the connection to the creating client is closed down if the close-down mode is **Destroy**. An **Access** error is generated if *counter* is a system counter. A **Counter** error is generated if *counter* does not name a valid counter.

QueryCounter

counter: COUNTER
⇒
value: INT64

Errors: **Counter**

This request returns the current value of the given counter or a generates **Counter** error if *counter* does not name a valid counter.

Await

wait-list: LISTofWAITCONDITION

Errors: **Counter**, **Alloc**, **Value**

When this request is executed, the triggers in the *wait-list* are initialized using the *wait-value* and *value-type* fields, as described in the definition of TRIGGER above. The processing of further requests for the client is blocked until one or more of the triggers becomes TRUE. This may happen immediately, as a result of the initialization, or at some later time, as a result of a subsequent **SetCounter**, **ChangeCounter** or **DestroyCounter** request.

A **Value** error is generated if *wait-list* is empty.

When the client becomes unblocked, each trigger is checked to determine whether a **CounterNotify** event should be generated. The difference between the *counter* and the test value is calculated by subtracting the test value from the value of the *counter*. If the *test-type* is **PositiveTransition** or **PositiveComparison**, a **CounterNotify** event is generated if the difference is at least *event-threshold*. If the *test-type* is **NegativeTransition** or **NegativeComparison**, a **CounterNotify** event is generated if the difference is at most *event-threshold*. If the difference lies outside the range for an INT64, an event is not generated.

This threshold check is made for each trigger in the list and a **CounterNotify** event is generated for every trigger for which the check succeeds. The check for **CounterNotify** events is performed even if one of the triggers is TRUE when the request is first executed. Note that a **CounterNotify** event may be generated for a *trigger* that is FALSE if there are multiple triggers in the request. A **CounterNotify** event with the *destroyed* flag set to TRUE is always generated if the counter for one of the triggers is destroyed.

ChangeCounter

counter: COUNTER
amount: INT64

Errors: **Counter**,**Access**,**Value**

This request changes the given counter by adding *amount* to the current counter value. If the change to this counter satisfies a trigger for which a client is waiting, that client is unblocked and one or more **CounterNotify** events may be generated. If the change to the counter satisfies the trigger for an alarm, an **AlarmNotify** event is generated and the alarm is updated. An **Access** error is generated if *counter* is a system counter. A **Counter** error is generated if *counter* does not name a valid counter. If the resulting value for the counter would be outside the range for an INT64, a **Value** error is generated and the counter is not changed.

It should be noted that *all* the clients whose triggers are satisfied by this change are unblocked, so this request cannot be used to implement mutual exclusion.

SetCounter

counter: COUNTER
value: INT64

Errors: **Counter**,**Access**

This request sets the value of the given counter to *value*. The effect is equivalent to executing the appropriate **ChangeCounter** request to change the counter value to *value*. An **Access** error is generated if *counter* names a system counter. A **Counter** error is generated if *counter* does not name a valid counter.

CreateAlarm

id: ALARM
values-mask: CARD32
values-list: LISTofVALUE

Errors: **IDChoice**,**Counter**,**Match**,**Value**,**Alloc**

This request creates an alarm and assigns the identifier *id* to it. The *values-mask* and *values-list* specify the attributes which are to be explicitly initialized. The attributes for an Alarm and their defaults are:

Attribute	Type	Default	
trigger	TRIGGER	counter	**None**
		value-type	**Absolute**
		value	0
		test-type	**PositiveComparison**
delta	INT64	1	
events	BOOL	TRUE	

The *trigger* is initialized as described in the definition of TRIGGER, with an error being generated if necessary.

If the *counter* is **None**, the state of the alarm is set to **Inactive**, else it is set to **Active**.

Whenever the *trigger* becomes TRUE, either as a result of this request or as the result of a **SetCounter**, **ChangeCounter**, **DestroyCounter** or **ChangeAlarm** request, an **AlarmNotify** event is generated and the alarm is updated. The alarm is updated by repeatedly adding *delta* to the *value* of the *trigger* and re-initializing it until it becomes FALSE. If this update would cause *value* to fall outside the range for an INT64, or if the *counter* value is **None**, or if the *delta* is 0 and *test-type* is **PositiveComparison** or **NegativeComparison**, no change is made to *value* and the alarm state is changed to **Inactive** before the event is generated. No further events are generated by an **Inactive** alarm until a **ChangeAlarm** or **DestroyAlarm** request is executed.

If the *test-type* is **PositiveComparison** or **PositiveTransition** and *delta* is less than zero, or if the *test-type* is **NegativeComparison** or **NegativeTransition** and *delta* is greater than zero, a **Match** error is generated.

The *events* value enables or disables delivery of **AlarmNotify** events to the requesting client. The alarm keeps a separate event flag for each client so that other clients may select to receive events from this alarm.

An **AlarmNotify** event is always generated at some time after the execution of a **CreateAlarm** request. This will happen immediately if the *trigger* is TRUE or it will happen later when the *trigger* becomes TRUE or the Alarm is destroyed.

ChangeAlarm

id: ALARM
values-mask: CARD32
values-list: LISTofVALUE

Errors: **Alarm**,**Counter**,**Value**,**Match**

This request changes the parameters of an Alarm. All of the parameters specified for the **CreateAlarm** request may be changed using this request. The *trigger* is re-initialized and an **AlarmNotify** event is generated if appropriate, as explained in the description of the **CreateAlarm** request.

Changes to the *events* flag affect the event delivery to the requesting client only, and may be used by a client to select or de-select event delivery from an alarm created by another client.

The order in which attributes are verified and altered is server-dependent. If an error is generated, a subset of the attributes may have been altered.

DestroyAlarm

alarm: ALARM

Errors: **Alarm**

This request destroys an alarm. An alarm is automatically destroyed when the creating client is closed down if the close-down mode is **Destroy**. When an alarm is destroyed, an **AlarmNotify** event is generated with a *state* value of **Destroyed**.

QueryAlarm

alarm: ALARM
⇒
trigger: TRIGGER
delta: INT64
events: ALARMEVENTMASK
state: ALARMSTATE

Errors: **Alarm**

This request retrieves the current parameters for an Alarm.

SetPriority

client-resource: XID
priority: INT32

Errors: **Match**

This request changes the scheduling priority of the client which created *client-resource*. If *client-resource* is **None** then the priority for the client making the request is changed. A **Match** error is generated if *client-resource* is not **None** and does not name an existing resource in the server. For any two *priority* values, `A` and `B`, `A` is higher priority if and only if `A` is greater than `B`.

The priority of a client is set to 0 when the initial client connection is made.

The effect of different client priorities depends on the particular implementation of the extension, and in some cases it may have no effect at all. However, the intention is that higher priority clients will have their requests executed before those of lower priority clients.

For most animation applications, it is desireable that animation clients are given priority over non-real-time clients. This improves the smoothness of the animation on a loaded server. Because a server is free to implement very strict priorities, processing requests for the highest priority client to the exclusion of all others, it is important that a client which may potentially monopolize the whole server, such as an

animation which produces continuous output as fast as it can with no rate control, is run at low rather than high priority.

GetPriority

client-resource: XID
⇒
priority: INT32

Errors: **Match**

This request returns the scheduling priority of the client which created *client-resource*. If *client-resource* is **None** then the priority for the client making the request is returned. A **Match** error is generated if *client-resource* is not **None** and does not name an existing resource in the server.

Events

CounterNotify

counter: COUNTER
wait-value: INT64
counter-value: INT64
time: TIME
count: CARD16
destroyed: BOOL

CounterNotify events may be generated when a client becomes unblocked after an **Await** request has been processed. *wait-value* is the value being waited for and *counter-value* is the actual value of the counter at the time the event was generated. The *destroyed* flag is TRUE if this request was generated as the result of the destruction of the counter, and FALSE otherwise. *time* is the server time at which the event was generated.

When a client is unblocked, all the **CounterNotify** events for the **Await** request are generated contiguously. If *count* is 0 there are no more events to follow for this request. If *count* is n, there are at least n more events to follow.

AlarmNotify

alarm: ALARM
counter-value: INT64
alarm-value: INT64
state: ALARMSTATE
time: TIME

An **AlarmNotify** event is generated when an alarm is triggered. *alarm-value* is the test value of the trigger in the alarm when it was triggered, *counter-value* is the value of the counter which triggered the alarm and *time* is the server time at which the event was generated. *state* is the new state of the alarm. If *state* is **Inactive**, no more events will be generated by this alarm until a **ChangeAlarm** request is executed, the alarm is destroyed or the counter for the alarm is destroyed.

Encoding

Please refer to the X11 Protocol Encoding document as this section uses syntactic conventions established there and references types defined there.

The name of this extension is "SYNC".

New Types

The following new types are used by the extension.

ALARM: CARD32
ALARMSTATE:

0	Active	
1	Inactive	
2	Destroyed	

COUNTER: CARD32
INT64: 64-bit signed integer
SYSTEMCOUNTER:

4	COUNTER	counter
8	INT64	resolution
2	n	length of name in bytes
n	STRING8	name
p		pad,p=pad(n+2)

TESTTYPE:

0	PositiveTransition
1	NegativeTransition
2	PositiveComparison
3	NegativeComparison

TRIGGER:

4	COUNTER	counter
4	VALUETYPE	wait-type
8	INT64	wait-value

4	TESTTYPE	test-type

VALUETYPE:

0	Absolute	
1	Relative	

WAITCONDITION:

20	TRIGGER	trigger
8	INT64	event threshold

An INT64 is encoded in 8 bytes with the most significant 4 bytes first followed by the least significant 4 bytes. Within these 4 byte groups, the byte ordering determined during connection setup is used.

Errors

Counter

1	0	Error
1	Base + 0	code
2	CARD16	sequence number
4	CARD32	bad counter
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Alarm

1	0	Error
1	Base + 1	code
2	CARD16	sequence number
4	CARD32	bad alarm
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Requests

Initialize

1	CARD8	major opcode
1	0	minor opcode
2	2	request length
1	CARD8	major version
1	CARD8	minor version
2		unused

⇒

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
1	CARD8	major version
1	CARD8	minor version
2		unused
20		unused

ListSystemCounters

1	CARD8	major opcode
1	1	minor opcode
2	1	request length

⇒

1	1	Reply
1		unused
2	CARD16	sequence number
4	*variable*	reply length
4	INT32	list length
20		unused
4n	list of SYSTEMCOUNTER	system counters

CreateCounter

1	CARD8	major opcode
1	2	minor opcode
2	4	request length
4	COUNTER	id
8	INT64	initial value

DestroyCounter

1	CARD8	major opcode
1	3	minor opcode
2	2	request length
4	COUNTER	counter

QueryCounter

1	CARD8	major opcode
1	4	minor opcode
2	2	request length
4	COUNTER	counter

⇒

1	1	Reply
1		unused

2	CARD16		sequence number
4	0		reply length
8	INT64		counter value
16			unused

Await

1	CARD8		major opcode
1	5		minor opcode
2	1 + 7*n		request length
28n	LISTofWAITCONDITION		wait conditions

ChangeCounter

1	CARD8		major opcode
1	6		minor opcode
2	4		request length
4	COUNTER		counter
8	INT64		amount

SetCounter

1	CARD8		major opcode
1	7		minor opcode
2	4		request length
4	COUNTER		counter
8	INT64		value

CreateAlarm

1	CARD8		major opcode
1	8		minor opcode
2	3+n		request length
4	ALARM		id
4	BITMASK		values mask
	#x00000001	counter	
	#x00000002	value-type	
	#x00000004	value	
	#x00000008	test-type	
	#x00000010	delta	
	#x00000020	events	
4n	LISTofVALUE		values

VALUES

4	COUNTER	counter
4	VALUETYPE	value-type
8	INT64	value
4	TESTTYPE	test-type
8	INT64	delta

4	BOOL	events

ChangeAlarm

1	CARD8	major opcode
1	9	minor opcode
2	3+n	request length
4	ALARM	id
4	BITMASK	values mask
	encodings as for **CreateAlarm**	
4n	LISTofVALUE	values
	encodings as for **CreateAlarm**	

DestroyAlarm

1	CARD8	major opcode
1	10	minor opcode
2	2	request length
4	ALARM	alarm

QueryAlarm

1	CARD8	major opcode
1	11	minor opcode
2	2	request length
4	ALARM	alarm

⇒

1	1	Reply
1		unused
2	CARD16	sequence number
4	2	reply length
20	TRIGGER	trigger
8	INT64	delta
1	BOOL	events
1	ALARMSTATE	state
2		unused

SetPriority

1	CARD8	major opcode
1	12	minor opcode
2	3	request length
4	CARD32	id
4	INT32	priority

GetPriority

1	CARD8	major opcode
1	13	minor opcode

2	1	request length
4	CARD32	id

⇒

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
4	INT32	priority
20		unused

Events

CounterNotify

1	Base + 0	code
1	0	kind
2	CARD16	sequence number
4	COUNTER	counter
8	INT64	wait value
8	INT64	counter value
4	TIME	timestamp
2	CARD16	count
1	BOOL	destroyed
1		unused

AlarmNotify

1	Base + 1	code
1	1	kind
2	CARD16	sequence number
4	ALARM	alarm
8	INT64	counter value
8	INT64	alarm value
4	TIME	timestamp
1	ALARMSTATE	state
3		unused

C Language Binding

The C routines provide direct access to the protocol and add no additional semantics.

The include file for this extension is `<X11/extensions/sync.h>`.

Most of the names in the language binding are derived from the protocol names by prepending XSync to the protocol name and changing the capitalization.

C Functions

Most of the following functions generate SYNC protocol requests.

`Status`
`XSyncQueryExtension` (`Display *` *dpy*, `int *` *event_base_return*,
    `int *` *error_base_return*)

If *dpy* supports the SYNC extension, this function returns `True`, sets **event_base_return* to the event number for the first SYNC event, and sets **error_base_return* to the error number for the first SYNC error. If *dpy* does not support the SYNC extension, this function returns `False`.

`Status`
`XSyncInitialize` (`Display *` *dpy*, `int *` *major_version_return*,
    `int *` *minor_version_return*)

Sets **major_version_return* and **minor_version_return* to the major/minor SYNC protocol version supported by the server. If the XSync library is compatible with the version returned by the server, this function returns `True`. If *dpy* does not support the SYNC extension, or if there was an error during communication with the server, or if the server and library protocol versions are incompatible, this function returns `False`. The only XSync function that may be called before this function is `XSyncQueryExtension`. If a client violates this rule, the effects of all XSync calls that it makes are undefined.

`XSyncSystemCounter *`
`XSyncListSystemCounters` (`Display *` *dpy*, `int *` *n_counters_return*)

Returns a pointer to an array of system counters supported by the display, and sets **n_counters_return* to the number of counters in the array. The array should be freed with `XSyncFreeSystemCounterList`. If *dpy* does not support the SYNC extension, or if there was an error during communication with the server, or if the server does not support any system counters, this function returns `NULL`.

`XSyncSystemCounter` has the following fields:

```
char *          name;       /* null-terminated name of system counter */
XSyncCounter    counter;    /* counter id of this system counter */
XSyncValue      resolution; /* resolution of this system counter */
```

`void`
`XSyncFreeSystemCounterList` (`XSyncSystemCounter *` *list*)

Frees the memory associated with the system counter list returned by `XSyncListSystemCounters`.

`XSyncCounter`
`XSyncCreateCounter` (`Display *` *dpy*, `XSyncValue` *initial_value*)

Creates a counter on the *dpy* with the given *initial_value* and returns the counter ID. Returns `None` if *dpy* does not support the SYNC extension.

`Status`
`XSyncSetCounter` (`Display *` *dpy*, `XSyncCounter` *counter*,
`XSyncValue` *value*)

Sets *counter* to *value*. Returns `False` if *dpy* does not support the SYNC extension, else returns `True`.

`Status`
`XSyncChangeCounter` (`Display *` *dpy*, `XSyncCounter` *counter*,
`XSyncValue` *value*)

Adds *value* to *counter*. Returns `False` if *dpy* does not support the SYNC extension, else returns `True`.

`Status`
`XSyncDestroyCounter` (`Display *` *dpy*, `XSyncCounter` *counter*)

Destroys *counter*. Returns `False` if *dpy* does not support the SYNC extension, else returns `True`.

`Status`
`XSyncQueryCounter` (`Display *` *dpy*, `XSyncCounter` *counter*,
`XSyncValue *` *value_return*)

Sets **value_return* to the current value of *counter*. Returns `False` if there was an error during communication with the server, or if *dpy* does not support the SYNC extension, else returns `True`.

`Status`
`XSyncAwait` (`Display *` *dpy*, `XSyncWaitCondition *` *wait_list*,
`int` *n_conditions*)

Awaits on the conditions in *wait_list*. *n_conditions* is the number of wait conditions in *wait_list*. Returns `False` if *dpy* does not support the SYNC extension, else returns `True`. The await is processed asynchronously by the server; this function always returns immediatley after issuing the request.

`XSyncWaitCondition` has the following fields:

```
XSyncCounter      trigger.counter;      /* counter to trigger on */
XSyncValueType    trigger.value_type;   /* absolute/relative */
XSyncValue        trigger.wait_value;   /* value to compare counter to */
XSyncTestType     trigger.test_type;    /* pos/neg comparison/transtion */
XSyncValue        event_threshold;      /* send event if past threshold */
```

`XSyncValueType` can be either `XSyncAbsolute` or `XSyncRelative`.

`XSyncTestType` can be one of `XSyncPositiveTransition`, `XSyncNegativeTransition`, `XSyncPositiveComparison`, or `XSyncNegativeComparison`.

`XSyncAlarm`
`XSyncCreateAlarm` (`Display *` *dpy*, `unsigned long` *values_mask*,
`XSyncAlarmAttributes *` *values*)

Creates an alarm and returns the alarm ID. Returns `None` if the display does not support the SYNC extension. *values_mask* and *values* specify the alarm attributes.

`XSyncAlarmAttributes` has the following fields. The attribute_mask column specifies the symbol that the caller should OR into *values_mask* to indicate that the value for the corresponding attribute was actually supplied. Default values are used for all attributes that do not have their attribute_mask OR'ed into *values_mask*. See the protocol description for CreateAlarm for the defaults.

type	field name	attribute_mask
`XSyncCounter`	trigger.counter;	`XSyncCACounter`
`XSyncValueType`	trigger.value_type;	`XSyncCAValueType`
`XSyncValue`	trigger.wait_value;	`XSyncCAValue`
`XSyncTestType`	trigger.test_type;	`XSyncCATestType`
`XSyncValue`	delta;	`XSyncCADelta`
`Bool`	events;	`XSyncCAEvents`
`XSyncAlarmState`	state;	client cannot set this

`Status`
`XSyncDestroyAlarm` (`Display *` *dpy*, `XSyncAlarm` *alarm*)

Destroys *alarm*. Returns `False` if *dpy* does not support the SYNC extension, else returns `True`.

```
Status
XSyncQueryAlarm ( Display * dpy, XSyncAlarm alarm,
    XSyncAlarmAttributes * values_return )
```

Sets **values_return* to the alarm's attributes. Returns `False` if there was an error during communication with the server, or if *dpy* does not support the SYNC extension, else returns `True`.

```
Status
XSyncChangeAlarm ( Display * dpy, XSyncAlarm alarm,
    unsigned long values_mask, XSyncAlarmAttributes * values )
```

Changes *alarm*'s attributes. The attributes to change are specified as in `XSyncCreateAlarm` Returns `False` if *dpy* does not support the SYNC extension, else returns `True`.

```
Status
XSyncSetPriority ( Display * dpy, XID client_resource_id, int priority )
```

Sets the priority of the client owning *client_resource_id* to *priority*. If *client_resource_id* is `None`, sets the caller's priority. Returns `False` if *dpy* does not support the SYNC extension, else returns `True`.

```
Status
XSyncGetPriority ( Display * dpy, XID client_resource_id,
    int * return_priority )
```

Sets **return_priority* to the priority of the client owning *client_resource_id*. If *client_resource_id* is `None`, sets **return_priority* to the caller's priority. Returns `False` if there was an error during communication with the server, or if *dpy* does not support the SYNC extension, else returns `True`.

C Macros/Functions

The following procedures manipulate 64 bit values. They are defined both as macros and as functions. By default, the macro form is used. To use the function form, #undef the macro name to uncover the function.

```
void
XSyncIntToValue ( XSyncValue *pv, int i )
```

Converts *i* to an `XSyncValue` and stores it in **pv*. Performs sign extension (**pv* will have the same sign as *i*.)

```
void
XSyncIntsToValue ( XSyncValue *pv, unsigned int low, int high )
```

Stores *low* in the low 32 bits of **pv* and *high* in the high 32 bits of **pv*.

`Bool`
`XSyncValueGreaterThan` (`XSyncValue` *a*, `XSyncValue` *b*)

Returns `True` if *a* is greater than *b*, else returns `False`.

`Bool`
`XSyncValueLessThan` (`XSyncValue` *a*, `XSyncValue` *b*)

Returns `True` if *a* is less than *b*, else returns `False`.

`Bool`
`XSyncValueGreaterOrEqual` (`XSyncValue` *a*, `XSyncValue` *b*)

Returns `True` if *a* is greater than or equal to *b*, else returns `False`.

`Bool`
`XSyncValueLessOrEqual` (`XSyncValue` *a*, `XSyncValue` *b*)

Returns `True` if *a* is less than or equal to *b*,

`Bool`
`XSyncValueEqual` (`XSyncValue` *a*, `XSyncValue` *b*)

Returns `True` if *a* is equal to *b*, else returns `False`.

`Bool`
`XSyncValueIsNegative` (`XSyncValue` *v*)

Returns `True` if *v* is negative, else returns `False`.

`Bool`
`XSyncValueIsZero` (`XSyncValue` *v*)

Returns `True` if *v* is zero, else returns `False`.

`Bool`
`XSyncValueIsPositive` (`XSyncValue` *v*)

Returns `True` if *v* is positive, else returns `False`.

`unsigned int`
`XSyncValueLow32` (`XSyncValue` *v*)

Returns the low 32 bits of *v*.

`int`
`XSyncValueHigh32` (`XSyncValue` *v*)

Returns the high 32 bits of *v*.

`void`
`XSyncValueAdd` (`XSyncValue *` *presult*, `XSyncValue` *a*, `XSyncValue` *b*, `Bool *` *poverflow*)

Adds *a* to *b* and stores the result in **presult*. If the result could not fit in 64 bits, **poverflow* is set to `True` else it is set to `False`.

`void`
`XSyncValueSubtract` (`XSyncValue *` *presult*, `XSyncValue` *a*, `XSyncValue` *b*, `Bool *` *poverflow*)

Subtracts *b* from *a* and stores the result in **presult*. If the result could not fit in 64 bits, *overflow* is set to `True` else it is set to `False`.

`void`
`XSyncMaxValue` (`XSyncValue *` *pv*)

Sets **pv* to the maximum value expressible in 64 bits.

`void`
`XSyncMinValue` (`XSyncValue *` *pv*)

Sets **pv* to the minimum value expressible in 64 bits.

Events

Let *event_base* be the value *event_base_return* as defined in the function `XSyncQueryExtension`.

An `XSyncCounterNotifyEvent`'s type field has the value *event_base* + `XSyncCounterNotify`. The fields of this structure are:

```
int             type;           /* event base + XSyncCounterNotify */
unsigned        long serial;    /* number of last request processed by server */
Bool            send_event;     /* true if this came from a SendEvent request */
Display *       display;        /* Display the event was read from */
XSyncCounter    counter;        /* counter involved in await */
XSyncValue      wait_value;     /* value being waited for */
XSyncValue      counter_value;  /* counter value when this event was sent */
Time            time;           /* milliseconds */
int             count;          /* how many more events to come */
Bool            destroyed;      /* True if counter was destroyed */
```

An `XSyncAlarmNotifyEvent`'s type field has the value *event_base* + `XSyncAlarmNotify`. The fields of this structure are:

```
int                type;           /* event base + XSyncAlarmNotify */
unsigned long      serial;         /* number of last request processed by server */
Bool               send_event;     /* true if this came from a SendEvent request */
Display *          display;        /* Display the event was read from */
XSyncAlarm         alarm;          /* alarm that triggered */
XSyncValue         counter_value;  /* value that triggered the alarm */
XSyncValue         alarm_value;    /* test value of trigger in alarm */
Time               time;           /* milliseconds */
XSyncAlarmState    state;          /* new state of alarm */
```

Errors

Let *error_base* be the value *error_base_return* as defined in the function `XSyncQueryExtension`.

An `XSyncAlarmError`'s error_code field has the value *error_base* + `XSyncBadAlarm`. The fields of this structure are:

```
int                type;
Display *          display;        /* Display the event was read from */
XSyncAlarm         alarm;          /* resource id */
unsigned long      serial;         /* serial number of failed request */
unsigned char      error_code;     /* error base + XSyncBadAlarm */
unsigned char      request_code;   /* Major op-code of failed request */
unsigned char      minor_code;     /* Minor op-code of failed request */
```

An `XSyncCounterError`'s error_code field has the value *error_base* + `XSyncBadCounter`. The fields of this structure are:

```
int                type;
Display *          display;        /* Display the event was read from */
XSyncCounter       counter;        /* resource id */
unsigned long      serial;         /* serial number of failed request */
unsigned char      error_code;     /* error base + XSyncBadCounter */
unsigned char      request_code;   /* Major op-code of failed request */
unsigned char      minor_code;     /* Minor op-code of failed request */
```