X Nonrectangular Window Shape Extension Protocol

X.Org Standard

Keith Packard, MIT X Consortium Intel Corporation
Hideki Hiura
SunSoft, Inc.

X Nonrectangular Window Shape Extension Protocol: X.Org Standard

by Keith Packard
Hideki Hiura
SunSoft, Inc.

Version 1.1

Copyright © 1989 The Open Group
Copyright © 2004 The Open Group
Copyright © 2006 Keith Packard

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 COPYRIGHT HOLDERS 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 copyright holders 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 copyright holders.

Table of Contents

Chapter 1. Overview

This extension provides arbitrary window and border shapes within the X11 protocol.

The restriction of rectangular windows within the X protocol is a significant limitation in the implementation of many styles of user interface. For example, many transient windows would like to display a "drop shadow'' to give the illusion of 3 dimensions. As another example, some user interface style guides call for buttons with rounded corners; the full simulation of a nonrectangular shape, particularly with respect to event distribution and cursor shape, is not possible within the core X protocol. As a final example, round clocks and nonrectangular icons are desirable visual addition to the desktop.

This extension provides mechanisms for changing both the visible and interactive shape of a window to arbitrary, possibly disjoint, nonrectangular forms. The intent of the extension is to supplement the existing semantics, not replace them. In particular, it is desirable for clients that are unaware of the extension to still be able to cope reasonably with shaped windows. For example, window managers should still be able to negotiate screen real estate in rectangular pieces. Toward this end, any shape specified for a window is clipped by the bounding rectangle for the window as specified by the window's geometry in the core protocol. An expected convention would be that client programs expand their shape to fill the area offered by the window manager.

Chapter 2. Description

Each window (even with no shapes specified) is defined by three regions: the *bounding region*, the *clip region* and the *input region*. The bounding region is the area of the parent window that the window will occupy (including border). The clip region is the subset of the bounding region that is available for subwindows and graphics. The area between the bounding region and the clip region is defined to be the border of the window. The input region is the subset of the bounding region that can "contain" the pointer.

A nonshaped window will have a bounding region that is a rectangle spanning the window, including its border; the clip region will be a rectangle filling the inside dimensions (not including the border); the input region will match the bounding region. In this document, these areas are referred to as the *default bounding region*, the *default clip region* and the *default input region*. For a window with inside size of *width* by *height* and border width *bwidth*, the default bounding, clip and input regions are the rectangles (relative to the window origin):

```
bounding.x = -bwidth
bounding.y = -bwidth
bounding.width = width + 2 * bwidth
bounding.height = height + 2 * bwidth

clip.x = 0
clip.y = 0
clip.width = width
clip.height = height

input.x = -bwidth
input.y = -bwidth
input.width = width + 2 * bwidth
input.height = height + 2 * bwidth
```

This extension allows a client to modify any combination of the bounding, clip or input regions by specifying new regions that combine with the default regions. These new regions are called the *client bounding region*, the *client clip region* and the *client input region*. They are specified relative to the origin of the window and are always defined by offsets relative to the window origin (that is, region adjustments are not required when the window is moved). Three mechanisms for specifying regions are provided: a list of rectangles, a bitmap, and an existing bounding or clip region from a window. This is modeled on the specification of regions in graphics contexts in the core protocol and allows a variety of different uses of the extension.

When using an existing window shape as an operand in specifying a new shape, the client region is used, unless none has been set, in which case the default region is used instead.

The *effective bounding region* of a window is defined to be the intersection of the client bounding region with the default bounding region. Any portion of the client bounding region that is not included in the default bounding region will not be included in the effective bounding region on the screen. This means that window managers (or other geometry managers) used to dealing with rectangular client windows will be able to constrain the client to a rectangular area of the screen.

Construction of the effective bounding region is dynamic; the client bounding region is not mutated to obtain the effective bounding region. If a client bounding region is specified that extends beyond the current default bounding region, and the window is later enlarged, the effective bounding region will be enlarged to include more of the client bounding region.

The *effective clip region* of a window is defined to be the intersection of the client clip region with both the default clip region and the client bounding region. Any portion of the client clip region that is not included in both the default clip region and the client bounding region will not be included in the effective clip region on the screen.

Construction of the effective clip region is dynamic; the client clip region is not mutated to obtain the effective clip region. If a client clip region is specified that extends beyond the current default clip region and the window or its bounding region is later enlarged, the effective clip region will be enlarged to include more of the client clip region if it is included in the effective bounding region.

The border of a window is defined to be the difference between the effective bounding region and the effective clip region. If this region is empty, no border is displayed. If this region is nonempty, the border is filled using the border-tile or border-pixel of the window as specified in the core protocol. Note that a window with a nonzero border width will never be able to draw beyond the default clip region of the window. Also note that a zero border width does not prevent a window from having a border, since the clip shape can still be made smaller than the bounding shape.

All output to the window and visible regions of any subwindows will be clipped to the effective clip region. The server must not retain window contents beyond the effective bounding region with backing store. The window's origin (for graphics operations, background tiling, and subwindow placement) is not affected by the existence of a bounding region or clip region.

The *effective input region* of a window is defined to be the intersection of the client input region with both the default input region and the client bounding region. Any portion of the client input region that is not included in both the default input region and the client bounding region will not be included in the effective input region on the screen.

Construction of the effective input region is dynamic; the client input region is not mutated to obtain the effective input region. If a client input region is specified that extends beyond the current default input region and the window or its bounding region is later enlarged, the effective input region will be enlarged to include more of the client input region if it is included in the effective bounding region.

Areas that are inside the default bounding region but outside the effective bounding region are not part of the window; these areas of the screen will be occupied by other windows. Input events that occur within the default bounding region but outside the effective bounding region will be delivered as if the window was not occluding the event position. Events that occur in a nonrectangular border of a window will be delivered to that window, just as for events that occur in a normal rectangular border.

An `InputOnly` window can have its bounding or input region set, but it is a `Match` error to attempt to set a clip region on an `InputOnly` window or to specify its clip region as a source to a request in this extension.

The server must accept changes to the clip and input regions of a root window, but the server is permitted to ignore requested changes to the bounding region of a root window. If the server accepts bounding region changes, the contents of the screen outside the bounding region are implementation dependent.

Chapter 3. Types

The following types are used in the request and event definitions in subsequent sections.

SHAPE_KIND: { `Bounding`, `Clip`, `Input` }

SHAPE_OP: { `Set`, `Union`, `Intersect`, `Subtract`, `Invert` }

`Set` indicates that the region specified as an explicit source in the request is stored unaltered as the new destination client region. `Union` indicates that the source and destination regions are unioned together to produce the new destination client region. `Intersect` indicates that the source and destination regions are intersected together to produce the new destination client region. `Subtract` indicates that the source region is subtracted from the destination region to produce the new destination region. `Invert` indicates that the destination region is subtracted from the source region to produce the new destination region.

Chapter 4. Requests

`ShapeQueryVersion`

=>

majorVersion: CARD16

minorVersion: CARD16

This request can be used to ensure that the server version of the SHAPE extension is usable by the client. This document defines major version one (1), minor version one (1).

`ShapeRectangles`

dest: WINDOW
destKind: SHAPE_KIND
op: SHAPE_OP
xOff, yOff: INT16
rectangles: LISTofRECTANGLES
ordering: { `UnSorted`, `YSorted`, `YXSorted`, `YXBanded` }
Errors: `Window`, `Length`, `Match`, `Value`

This request specifies an array of rectangles, relative to the origin of the window plus the specified offset (xOff and yOff) that together define a region. This region is combined (as specified by the operator op) with the existing client region (specified by destKind) of the destination window, and the result is stored as the specified client region of the destination window. Note that the list of rectangles can be empty, specifying an empty region; this is not the same as passing `None` to `ShapeMask`,

If known by the client, ordering relations on the rectangles can be specified with the ordering argument. This may provide faster operation by the server. The meanings of the ordering values are the same as in the core protocol `SetClipRectangles` request. If an incorrect ordering is specified, the server may generate a `Match` error, but it is not required to do so. If no error is generated, the graphics results are undefined. Except for `UnSorted` , the rectangles should be nonintersecting, or the resulting region will be undefined. `UnSorted` means that the rectangles are in arbitrary order. `YSorted` means that the rectangles are nondecreasing in their Y origin. `YXSorted` additionally constrains `YSorted` order in that all rectangles with an equal Y origin are nondecreasing in their X origin. `YXBanded` additionally constrains `YXSorted` by requiring that, for every possible Y scanline, all rectangles that include that scanline have identical Y origins and Y extents.

`ShapeMask`

dest: WINDOW *destKind*: SHAPE_KIND *op*: SHAPE_OP *xOff, yOff*: INT16 *source*: PIXMAP or `None`
Errors: `Window`, `Pixmap`, `Match`, `Value`

The source in this request is a 1-bit deep pixmap, or `None` . If source is `None` , the specified client region is removed from the window, causing the effective region to revert to the default region. The `ShapeNotify` event generated by this request and subsequent `ShapeQueryExtents` will report that a client shape has not been specified. If a valid pixmap is specified, it is converted to a region, with bits set to one included in the region and bits set to zero excluded, and an offset from the window origin as specified by xOff and yOff. The resulting region is then combined (as specified by the operator op) with the existing client region (indicated by destKind) of the destination window, and the result is stored as the specified client region of the destination window. The source pixmap and destination window must have been created on the same screen, or else a `Match` error results.

`ShapeCombine`

dest: WINDOW

destKind: SHAPE_KIND

op: SHAPE_OP

xOff, yOff: INT16

source: WINDOW

sourceKind: SHAPE_KIND

Errors: `Window`, `Match`, `Value`

The client region, indicated by sourceKind, of the source window is offset from the window origin by xOff and yOff and combined with the client region, indicated by destKind, of the destination window. The result is stored as the specified client region of the destination window. The source and destination windows must be on the same screen, or else a `Match` error results.

`ShapeOffset`

dest: WINDOW

destKind: SHAPE_KIND

xOff, yOff: INT16

Errors: `Window`, `Match`, `Value`

The client region, indicated by destKind, is moved relative to its current position by the amounts xOff and yOff.

`ShapeQueryExtents`

dest: WINDOW

=>

boundingShaped: BOOL

clipShaped: BOOL

xBoundingShape: INT16

yBoundingShape: INT16

widthBoundingShape: CARD16

heightBoundingShape: CARD16

xClipShape: INT16

yClipShape: INT16

widthClipShape: CARD16

heightClipShape: CARD16

Errors: `Window`

The boundingShaped and clipShaped results are `True` if the corresponding client regions have been specified, else they are `False` . The x, y, width, and height values define the extents of the client regions, when a client region has not been specified, the extents of the corresponding default region are reported.

`ShapeSelectInput`

window: WINDOW

enable: BOOL

Errors: `Window`, `Value`

Specifying enable as `True` causes the server to send the requesting client a `ShapeNotify` event whenever the bounding, clip or input region of the specified window is altered by any client. Specifying enable as `False` causes the server to stop sending such events.

`ShapeInputSelected`

window: WINDOW

=> enable: BOOL

Errors: `Window`

If enable is `True`, then `ShapeNotify` events for the window are generated for this client.

`ShapeGetRectangles`

window: WINDOW

kind: SHAPE_KIND

=> rectangles: LISTofRECTANGLE

ordering: { `UnSorted`, `YSorted`, `YXSorted`, `YXBanded` }

Errors: `Window`, `Match`

A list of rectangles describing the region indicated by kind, and the ordering of those rectangles, is returned. The meaning of the ordering values is the same as in the `ShapeRectangles` request.

Chapter 5. Events

`ShapeNotify`

window: WINDOW
kind: SHAPE_KIND
shaped: BOOL
x, y: INT16
width, *height*: CARD16
time: TIMESTAMP

Whenever the client bounding, clip or input shape of a window is modified, a `ShapeNotify` event is sent to each client that has used `ShapeSelectInput` to request it.

Kind indicates which client region (bounding or clip) has been modified; shaped is `True` when the window has a client shape of type kind, and is `False` when the window no longer has a client shape of this type. The x, y, width, and height indicate the extents of the current shape. When shaped is `False` these will indicate the extents of the default region. The timestamp indicates the server time when the shape was changed.

Chapter 6. Encoding

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

The name of this extension is "SHAPE".

New Types

```
SHAPE_KIND
     0     Bounding
     1     Clip
     2     Input
```

```
SHAPE_OP
     0     Set
     1     Union
     2     Intersect
     3     Subtract
     4     Invert
```

Requests

```
ShapeQueryVersion
     1     CARD8                  opcode
     1     0                      shape opcode
     2     1                      request length

=>
     1     1                      Reply
     1                            unused
     2     CARD16                 sequence number
     4     0                      length
     2     CARD16                 major version
     2     CARD16                 minor version
     20                           unused
```

```
ShapeRectangles
     1     CARD8                  opcode
     1     1                      shape opcode
     2     4+2n                   request length
     1     SHAPE_OP               operation
     1     SHAPE_KIND             destination kind
     1                            ordering
           0     UnSorted
           1     YSorted
           2     YXSorted
           3     YXBanded
```

```
     1                           unused
     4     WINDOW                destination window
     2     INT16                 x offset
     2     INT16                 y offset
     8n    LISTofRECTANGLE       rectangles


ShapeMask
     1     CARD8                 opcode
     1     2                     shape opcode
     2     5                     request length
     1     SHAPE_OP              operation
     1     SHAPE_KIND            destination kind
     2                           unused
     4     WINDOW                destination window
     2     INT16                 x offset
     2     INT16                 y offset
     4     PIXMAP                source bitmap
           0     None


ShapeCombine
     1     CARD8                 opcode
     1     3                     shape opcode
     2     5                     request length
     1     SHAPE_OP              operation
     1     SHAPE_KIND            destination kind
     1     SHAPE_KIND            source kind
     1                           unused
     4     WINDOW                destination window
     2     INT16                 x offset
     2     INT16                 y offset
     4     WINDOW                source window


ShapeOffset
     1     CARD8                 opcode
     1     4                     shape opcode
     2     4                     request length
     1     SHAPE_KIND            destination kind
     3                           unused
     4     WINDOW                destination window
     2     INT16                 x offset
     2     INT16                 y offset


ShapeQueryExtents
     1     CARD8                 opcode
     1     5                     shape opcode
     2     2                     request length
     4     WINDOW                destination window

 =>
     1     1                     Reply
     1                           unused
```

```
     2     CARD16              sequence number
     4     0                   reply length
     1     BOOL                bounding shaped
     1     BOOL                clip shaped
     2                         unused
     2     INT16               bounding shape extents x
     2     INT16               bounding shape extents y
     2     CARD16              bounding shape extents width
     2     CARD16              bounding shape extents height
     2     INT16               clip shape extents x
     2     INT16               clip shape extents y
     2     CARD16              clip shape extents width
     2     CARD16              clip shape extents height
     4                         unused


ShapeSelectInput
     1     CARD8               opcode
     1     6                   shape opcode
     2     3                   request length
     4     WINDOW              destination window
     1     BOOL                enable
     3                         unused


ShapeInputSelected
     1     CARD8               opcode
     1     7                   shape opcode
     2     2                   request length
     4     WINDOW              destination window
 =>
     1     1                   Reply
     1     BOOL                enabled
     2     CARD16              sequence number
     4     0                   reply length
     24                        unused


ShapeGetRectangles
     1     CARD8               opcode
     1     8                   shape opcode
     2     3                   request length
     4     WINDOW              window
     1     SHAPE_KIND          source kind
     3                         unused
 =>
     1     1                   Reply
     1                         ordering
           0     UnSorted
           1     YSorted
           2     YXSorted
           3     YXBanded
     2     CARD16              sequence number
     4     2n                  reply length
```

```
     4     CARD32             nrects
     20                       unused
     8n    LISTofRECTANGLE    rectangles
```

Events

```
ShapeNotify
     1     CARD8               type (0 + extension event base)
     1     SHAPE_KIND          shape kind
     2     CARD16              sequence number
     4     WINDOW              affected window
     2     INT16               x value of extents
     2     INT16               y value of extents
     2     CARD16              width of extents
     2     CARD16              height of extents
     4     TIMESTAMP           server time
     1     BOOL                shaped
     11                        unused
```

Glossary

bounding region	The area of the parent window that this window will occupy. This area is divided into two parts: the border and the interior.
clip region	The interior of the window, as a subset of the bounding region. This region describes the area that will be painted with the window background when the window is cleared, will contain all graphics output to the window, and will clip any subwindows.
input region	The subset of the bounding region which can ``contain'' the pointer.
default bounding region	The rectangular area, as described by the core protocol window size, that covers the interior of the window and its border.
default clip region	The rectangular area, as described by the core protocol window size, that covers the interior of the window and excludes the border.
default input region	The rectangular area, as described by the core protocol window size, that covers the interior of the window and its border.
client bounding region	The region associated with a window that is directly modified via this extension when specified by `ShapeBounding` . This region is used in conjunction with the default bounding region to produce the effective bounding region.
client clip region	The region associated with a window that is directly modified via this extension when specified by `ShapeClip` . This region is used in conjunction with the default clip region and the client bounding region to produce the effective clip region.
client input region	The region associated with a window that is directly modified via this extension when specified by `ShapeInput` . This region is used in conjunction with the default input region and the client bounding region to produce the effective input region.
effective bounding region	The actual shape of the window on the screen, including border and interior (but excluding the effects of overlapping windows). When a window has a client bounding region, the effective bounding region is the intersection of the default bounding region and the client bounding region. Otherwise, the effective bounding region is the same as the default bounding region.
effective clip region	The actual shape of the interior of the window on the screen (excluding the effects of overlapping windows). When a window has a client clip region or a client bounding region, the effective clip region is the intersection of the default clip region, the client clip region (if any) and the client bounding region (if any). Otherwise, the effective clip region is the same as the default clip region.
effective input region	The actual shape of the window on the screen (excluding the effects of overlapping windows) which can ``contain'' the pointer. When a window has a client input region or a client bounding

region, the effective input region is the intersection of the default input region, the client input region (if any) and the client bounding region (if any). Otherwise, the effective input region is the same as the default input region.