The *destination* vector is created automatically by CLX. *destination* is guaranteed to have room for (– *source-end source-star*t) integer elements, starting at *destination-start*. Elements of *destination* can be either **card8** or **card16** integers, depending on the context. *font* is the current font, if known, or **nil** otherwise. Starting with the element at *source-start*, **translate-function** should translate as many elements of *source* as possible (up to the *source-end* element) into indexes in the current *font*, and store them into *destination*. The first return value should be the source index of the first untranslated element.

The second return value indicates the changes which should be made to the current text output request before translating the remaining *source* elements. If no further elements need to be translated, the second return value should be **nil**. If a horizontal motion is required before further translation, the second return value should be the change in x position. If a font change is required for further translation, the second return value should be the new font.

If known, the pixel width of the translated text can be returned as the third value; this can allow for appending of subsequent output to the same protocol request, if no overall width has been specified at the higher level.

*source* — A sequence of characters or other objects to be translated.

*source-start* — An array-index specifying the first *source* element to be translated.

*source-end* — An array-index specifying the end of the *source* subsequence to be translated.

*font* — The font indexed by translated *source* elements.

*destination* — A vector where translated *source* elements are stored.

*destination-start* — An array-index specifying the position to begin storing translated *source* elements.

**draw-image-glyphs** *drawable gcontext x y sequence* &key (**:start** 0) **:end** **:translate :width** (**:size :default**) Function

Returns:
*new-start* — Type **array-index** or **null**.
*width* — Type **int32** or **null**.

Draws the block text characters represented by the given **sequence**. **:start** and **:end** define the elements of the *sequence* which are drawn. The given *x* and *y* specify the left baseline position for the first character. The first return value is **nil** if all characters are successfully translated and drawn; otherwise, the index of the first untranslated sequence element is returned. The **:translate** function is allowed to return an initial font change. The second return value gives the total pixel width of the characters actually drawn, if known.

The **:translate** function may not return a horizontal position change, since **draw-image-glyphs** does not generate complex output requests.

Specifying a **:width** is a hint to improve performance. The **:width** is assumed to be the total pixel width of the character sequence actually drawn. Specifying **:width** permits appending the output of subsequent calls to the same protocol request, provided *gcontext* has not been modified in the interim. If **:width** is not specified, appending of subsequent output might not occur (unless **:translate** returns the character width).

The **:size** specifies the element size of the destination buffer given to **:translate** (either 8, 16, or **:default**). If **:default** is specified, the size will be based on the current font, if known; otherwise, 16 is used.

*drawable* — The destination **drawable**.

*x*, *y* — The left baseline position for the character drawn.

*gcontext* — The graphics context for drawing text.

*sequence* — A sequence of characters or other objects to be translated into font indexes.

**:start**, **:end** — Start and end indexes defining the elements to draw.

**:translate** — A function to translate text to font indexes. Default is **#'translate-default**.

**:width** — The total total pixel width of the character actually drawn, if known.

**:size** — The element size of the destination buffer given to **:translate** (8, 16, or **:default**).

**translate-function** *source source-start source-end font destination destination-start* Function

Returns:
*first-not-done* — Type **array-index**.
*to-continue* — Type **int16**, **font**, or **null**.
*current-width* — Type **int32** or **null**.

A function used as the **:translate** argument for text functions. Converts elements of the *source* (sub)sequence into font indexes for the given *font* and stores them into the *destination* vector.

*gcontext* — The graphics context for drawing text.

*x*, *y* — The left baseline position for the character drawn.

*sequence* — A sequence of characters or other objects to be translated into font indexes.

**:start**, **:end** — Start and end indexes defining the elements to draw.

**:translate** — A function to translate text to font indexes. Default is **#'translate-default**.

**:width** — The total total pixel width of the character actually drawn, if known.

**:size** — The element size of the destination buffer given to **:translate** (8, 16, or **:default**).

**draw-image-glyph** *drawable gcontext x y element* &key **:translate :width** (**:size :default**) Function

Returns:
*output-p* — Type **boolean**.
*width* — Type **int32** or **null**.

Draws a single character of block text represented by the given *element*. The given *x* and *y* specify the left baseline position for the character. The first return value is true if the character is successfully translated and drawn, or **nil** if the **:translate** function did not translate it. The **:translate** function is allowed to return an initial font change. The second return value gives the total pixel width of the character actually drawn, if known.

The **:translate** function may not return a horizontal position change, since **draw-image-glyph** does not generate complex output requests.

Specifying a **:width** is a hint to improve performance. The **:width** is assumed to be the total pixel width of the character actually drawn. Specifying **:width** permits appending the output of subsequent calls to the same protocol request, provided *gcontext* has not been modified in the interim. If **:width** is not specified, appending of subsequent output might not occur (unless **:translate** returns the character width).

The **:size** specifies the element size of the destination buffer given to **:translate** (either 8, 16, or **:default**). If **:default** is specified, the size is based on the current font, if known; otherwise, 16 is used.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing text.

*x*, *y* — The left baseline position for the character drawn.

*element* — A character or other object to be translated into a font index.

**:translate** — A function to translate text to font indexes. Default is **#'translate-default**.

**:width** — The total pixel width of the character actually drawn, if known.

**:size** — Specifies the element size of the destination buffer given to **:translate** (8, 16, or **:default**).

**draw-glyph** *drawable gcontext x y element* &key **:translate :width** Function
(**:size :default**)
Returns:
*output-p* — Type **boolean**.
*width* — Type **int32** or **null**.

Draws a single character of filled text represented by the given *element*. The given *x* and *y* specify the left baseline position for the character. The first return value is true if the character is successfully translated and drawn, or **nil** if the **:translate** function did not translate it. The second return value gives the total pixel width of the character actually drawn, if known.

Specifying a **:width** is a hint to improve performance. The **:width** is assumed to be the total pixel width of the character actually drawn. Specifying **:width** permits appending the output of subsequent calls to the same protocol request, provided *gcontext* has not been modified in the interim. If **:width** is not specified, appending of subsequent output might not occur (unless **:translate** returns the character width).

The **:size** specifies the element size of the destination buffer given to **:translate** (either 8, 16, or **:default**). If **:default** is specified, the size is based on the current font, if known; otherwise, 16 is used.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing text.

*x*, *y* — The left baseline position for the character drawn.

*element* — A character or other object to be translated into a font index.

**:translate** — A function to translate text to font indexes. Default is **#'translate-default**.

**:width** — The total pixel width of the character actually drawn, if known.

**:size** — Specifies the element size of the destination buffer given to **:translate** (8, 16, or **:default**).

**draw-glyphs** *drawable gcontext x y sequence* &key (**:start** 0) **:end :translate** Function
**:width** (**:size :default**)
Returns:
*new-start* — Type **array-index** or **null**.
*width* — Type **int32** or **null**.

Draws the filled text characters represented by the given sequence. **:start** and **:end** define the elements of the sequence which are drawn. The given *x* and *y* specify the left baseline position for the first character. The first return value is **nil** if all characters are successfully translated and drawn; otherwise, the index of the first untranslated sequence element is returned. The second return value gives the total pixel width of the characters actually drawn, if known.

Specifying a **:width** is a hint to improve performance. The **:width** is assumed to be the total pixel width of the character sequence actually drawn. Specifying **:width** permits appending the output of subsequent calls to the same protocol request, provided *gcontext* has not been modified in the interim. If **:width** is not specified, appending of subsequent output might not occur (unless **:translate** returns the character width).

The **:size** specifies the element size of the destination buffer given to **:translate** (either 8, 16, or **:default**). If :**default** is specified, the size is based on the current font, if known; otherwise, 16 is used.

*drawable* — The destination **drawable**.

**draw-arcs** *drawable gcontext arcs* &optional *fill-p* Function

Draws circular or elliptical, outlined or filled arcs. Each arc is specified by a rectangle and two angles. For a more detailed description, see **draw-arc**.

The arcs are filled in the order listed. For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn multiple times.

*drawable* — Specifies the **drawable** where you want the arcs drawn.

*gcontext* — Specifies the graphics context for drawing the arc.

*arcs* — A sequence containing the width, height, angle1, and angle2 arguments defining the arcs. See **draw-arc** for more detail. Type is **arc-seq**.

*fill-p* — Specifies whether the arcs are filled or not. Type is **boolean**.

## Drawing Text

**6.7** CLX provides functions for drawing text using text fonts provided by the X server. An X font is array of character bit maps indexed by integer codes. See Section 8 for a complete discussion of the CLX functions used to manage fonts and characters.

Since Common Lisp programs typically represent text as sequences of characters (that is, strings), CLX text functions must be prepared to convert a Common Lisp character into the integer code used to index the appropriate character bitmap in a given font. The **:translate** argument to a text function is a function which performs this conversion. The default **:translate** function handles all characters that satisfy **graphic-char-p** by converting each character into its ASCII code. Note that the assumption made by the default **:translate** function—that is, that an X font indexes bitmaps by ASCII codes—is often valid, but other encodings are possible. In general, a **:translate** function can perform complex transformations. It can be used to convert non-character input, to handle non-ASCII character encodings, and to change the fonts used to access character bitmaps. The complete behavior of a **:translate** function is given below by describing a prototypical **translate-function**.

CLX offers two different ways to draw text—filled text and block text. The **draw-glyph** and **draw-glyphs** functions create filled text, in which each character image is treated as an area to be filled according to the fill-style of the given graphics context, without otherwise disturbing the surrounding background. In addition, filled text sends a complex type of server request which allows a series of font indices, font changes, and horizontal position changes to be compiled into a single request. Filled text functions use the following graphics context attributes: background, clip-mask, clip-x-origin, clip-y-origin, fill-style, font, foreground, function, plane-mask, stipple, subwindow-mode, tile, ts-x-origin, ts-y-origin.

Block text is a rendering style commonly used by display terminals, in which each character image appears in the foreground pixel inside a rectangular character cell drawn in the graphics context background pixel. The **draw-image-glyph** and **draw-image-glyphs** functions create block text. Block text functions use the following graphics context attributes: background, clip-mask, clip-x-origin, clip-y-origin, font, foreground, plane-mask, stipple, subwindow-mode, tile, ts-x-origin, ts-y-origin.

## Drawing Arcs

**6.6** **draw-arc** draws a single circular or an elliptical arc, while **draw-arcs** draws multiple circular or elliptical arcs. **draw-arc** and **draw-arcs** use the following graphics context components: arc-mode, background, cap-style, clip-x, clip-y, clip-mask, dash-list, dash-offset, fill-style, foreground, join-style, function, plane-mask, line-width, line-style, stipple, subwindow-mode, tile, ts-x-origin, and ts-y-origin.

**draw-arc** *drawable gcontext x y width height angle1 angle2* &optional *fill-p* Function

Draws either a circular or an elliptical arc. Also, outlined or filled arcs can be drawn. Each arc is specified by a rectangle (*x*, *y*, *width*, and *height*) and two angles (*angle1* and *angle2*). The angles are signed integers in radians, with positive indicating counter-clockwise motion and negative indicating clockwise motion. The start of the arc is specified by *angle1*, and the path and extent of the arc is specified by *angle2* relative to the start of the arc. If the magnitude of *angle2* is greater than 360 degrees, it is truncated to 360 degrees. The *x* and *y* coordinates of the rectangle are relative to the *drawable*'s origin.

For example, an arc specified as [*x*,*y*,*width*,*height*,*angle1*,*angle2*] has the origin of the major and minor axes at:

[*x*+(*width*/2),*y*+(*height*/2)]

The infinitely thin path describing the entire circle/ellipse intersects the horizontal axis at:

[*x*,*y*+(*height*/2)] and [*x*+*width*,*y*+(*height*/2)]

The intersection of the vertical axis is at:

[*x*+(*width*/2),*y*] and [*x*+(*width*/2),*y*+*height*]

These coordinates can be fractional; that is, they are not truncated to discrete coordinates. Note that the angle values are slightly different in CLX than in the X protocol specification.

If *fill-p* is **nil**, then only the outline of the arc is drawn. Otherwise, if *fill-p* is true, **draw-arc** fills the area bounded by the arc outline and one or two line segments, depending on the arc-mode. If the arc-mode is **:chord**, the filled area is bounded by the arc outline and the line segment joining the arc end points. If the arc-mode is **:pie-slice**, the filled area is bounded by the arc outline and the two line segments joining each arc end point with the center point.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing the arc.

*x*, *y* — The x and y coordinates of the arc rectangle relative to the origin of the *drawable*. Type is **int16**.

*width*, *height* — Specifies the width and height of the rectangle. These are the major and minor axes of the arc. Type is **card16**.

*angle1* — Specifies the start of the arc in radians. Type is **angle**.

*angle2* — Specifies the direction and end point of the arc. Type is **angle**.

*fill-p* — Specifies whether the arc is filled or not. Type **boolean**.

**:fill-p** — When true, a filled polygon is drawn instead of a polyline.

**:shape** — A hint that allows the server to use the most efficient area fill algorithm. Either **:convex**, **:non-convex**, or **:complex**.

**draw-segments** *drawable gcontext segments* Function

Draws multiple lines, not necessarily connected. *segments* is a sequence of the form {x1 y1 x2 y2}*, in which each subsequence specifies the end points of a line segment. Line segments are drawn in the order given by *segments*. Unlike **draw-lines**, no joining is performed at coincident end points.

*drawable* — The destination **drawable** to receive the line segments.

*gcontext* — Specifies the graphics context for drawing the lines.

*segments* — The points list for the segments to draw. Type is **seq**.

## Drawing Rectangles

**6.5** The **draw-rectangle** and **draw-rectangles** functions draw hollow or filled outlines of the specified rectangle or rectangles as if a five-point polyline were specified for each rectangle, as follows:

[x,y,] [x+width,y] [x+width,y+height] [x,y+height] [x,y]

**draw-rectangle** and **draw-rectangles** use the following graphics context components: background, function, plane-mask, foreground, subwindow-mode, cap-style, clip-x, clip-y, clip-ordering, clip-region and clip-mask, dash-list, dash-offset, fill-style, join-style, line-width, line-style, stipple, tile, ts-x-origin, and ts-y-origin.

**draw-rectangle** *drawable gcontext x y width height* &optional *fill-p* Function

Draws a rectangle defined by the *x*, *y*, *width*, and *height* arguments.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing the rectangle.

*x*, *y* — The x and y coordinates that define the upper left corner of the rectangle. The coordinates are relative to the destination origin. Type is **int16**.

*width*, *height* — Specifies the width and height that define the outline of the rectangle. Type is **card16**.

*fill-p* — Specifies whether the rectangle is filled or not. Type **boolean**.

**draw-rectangles** *drawable gcontext rectangles* &optional *fill-p* Function

Draws the rectangles in the order listed in *rectangles*. For the specified *rectangle* or *rectangles*, no pixel is drawn more than once. The x and y coordinates of each rectangle are relative to the destination origin and define the upper left corner of the rectangle. If rectangles intersect, the intersecting pixels are drawn multiple times.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context.

*rectangles* — A list specifying the upper left corner x and y, width and height of the rectangles. Type is **rect-seq**.

*fill-p* — Specified if the rectangles are filled or not. Type is **boolean**.

The **draw-lines** function also uses the join-style graphics context component.

**draw-line** *drawable gcontext x1 y1 x2 y2* &optional *relative-p* Function

Draws a line from the point *x1*,*y1* to the point *x2*,*y2*. When *relative-p* is true, the first point is relative to the destination origin but the second point is relative to the first point. When *relative-p* is **nil**, both points are relative to the destination origin.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing the line.

*x1*, *y1*, *x2*, *y2* — The end points of the line.

*relative-p* — Specifies the coordinate mode used for drawing the line either relative to the origin or the previous point. In either case, the first point is always drawn relative to the *drawable*'s origin.

**draw-lines** *drawable gcontext points* &key **:relative-p :fill-p** (**:shape :complex**) Function

Draws a line between each pair of *points* in the points list. The lines are drawn in the order listed and join correctly at all intermediate points. The join-style graphics context component defines the type of joint to use. When the first and last points coincide, the first and last lines also join correctly to produce a hollow polygon.

When **:relative-p** is true, the first point is always relative to the destination origin, but the rest are relative to the previous point. When **:relative-p** is **nil**, the rest of the points are drawn relative to the destination origin.

When **:fill-p** is true, the polygon defined by the *points* list is filled. The **:shape** keyword provides the server with a hint about how to fill the polygon. **:shape** can be either **:complex** (by default), **:convex**, or **:non-convex**.

The **:convex** operand is the simplest type of area and the fastest to fill. A fill area is convex if every straight line connecting any two interior points is entirely inside the area. For example, triangles and rectangles are convex polygons.

The **:non-convex** operand is for filling an area that is not convex and is also not self-intersecting. Filling this type of area is harder than filling a convex area, but easier than filling one that is self-intersecting. For example, the shape of the letter "T" is non-convex and non-self-intersecting.

The **:complex** operand is the most general (and therefore the hardest) type of fill area. A complex fill area can be non-convex and self-intersecting. For example, draw the outline of a bow tie, without lifting your pencil or tracing over an edge twice. This shape is non-convex and intersects itself at the knot in the middle.

---

**NOTE:** Unless you are sure that a shape is **:convex** or **:non-convex**, it should always be drawn as a **:complex** shape. If **:convex** or **:non-convex** is specified incorrectly, the graphics result is undefined.

---

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing the lines.

*points* — A list of points that define the lines. Type is **point-seq**.

**:relative-p** — The coordinate mode of the points.

*plane* — Specifies the bit-plane of the *source* **drawable**. Exactly one bit must be set. Type is **pixel**.

*source-x*, *source-y* — The *x* and *y* coordinates of the upper-left corner of the area in the *source* **drawable**. These coordinates are relative to the *source* **drawable** origin. Type is **int16**.

*width*, *height* — The width and height of the area being copied. These apply to both the *source* and *destination* areas. Type is **card16**.

*destination* — The destination **drawable**.

*destination-x*, *destination-y* — The x and y coordinates of the upper-left corner of the destination area in the *destination* **drawable**. These coordinates are relative to the *destination* **drawable** origin. Type is **int16**.

## Drawing Points

**6.3** The **draw-point** and **draw-points** functions make use of the following graphics context components: function, plane-mask, foreground, subwindow-mode, clip-x, clip-y, clip-ordering, clip-region and clip-mask.

The **draw-point** function uses the foreground pixel and function components of the graphics context to draw a single point into the specified drawable, while **draw-points** draws multiple points into the specified drawable. These functions are not affected by the tile or stipple in the graphics context.

**draw-point** *drawable gcontext x y* Function

Combines the foreground pixel in the *gcontext* with the pixel in the *drawable* specified by the *x* and *y* coordinates.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing the point.

*x*, *y* — The *x* and *y* coordinates of the point drawn. Type is **int16**.

**draw-points** *drawable gcontext points* &optional *relative-p* Function

Combines the foreground pixels in the graphics context with the pixels at each point in the *drawable*. The points are drawn in the order listed.

**draw-points** requires a mode argument, *relative-p* that indicates whether the points are relative to the destination origin or to the previous point. In either case, the first point is always relative to the destination origin. The rest of the points are relative either to the *drawable*'s origin or to the previous point, depending on the value of *relative-p*.

*drawable* — The destination **drawable**.

*gcontext* — The graphics context for drawing the points.

*points* — A list of points to be drawn in the order listed. The first point is always relative to the *drawable*'s origin; if *relative-p*, the rest of the points are drawn relative to the previous point, else they are drawn relative to the *drawable*'s origin. Type is **point-seq**.

*relative-p* — Specifies the coordinate mode used for drawing the pixels either relative to the origin or to the previous point. Type **boolean**.

## Drawing Lines

**6.4** The **draw-line**, **draw-lines**, and **draw-segments** functions use the following graphics context components: background, cap-style, clip-x-origin, clip-y-origin, clip-mask, dash-list, dash-offset, fill-style, foreground, function, plane-mask, line-width, line-style, stipple, subwindow-mode, tile, ts-x-origin, and ts-y-origin.

**copy-area** *source gcontext source-x source-y width height destination destination-x destination-y* Function

Copies the specified rectangular area from the *source* **drawable** to the specified rectangular area of the *destination* **drawable**, combining them as specified in the supplied graphics context (*gcontext*). The *x* and *y* coordinates are relative to their respective drawable origin, with each pair specifying the upper left corner of the area.

If either regions of the *source* area are obscured and have not been retained in backing store, or regions outside the boundaries of the *source* **drawable** are specified, those regions are not copied. Instead, the following occurs on all corresponding *destination* regions that are either visible or are retained in backing store:

- If the *destination* rectangle is a window with a background other than **:none**, these corresponding regions of the *destination* are tiled, using plane-mask of all ones and function of **boole-1** (copy source), with that background.
- If the exposures attribute of the graphics context is **:on**, then **:graphics-exposure** events for all corresponding *destination* regions are generated (regardless of tiling or whether the *destination* is a window or a pixmap).
- If exposures is **:on** but no regions are exposed, a **:no-exposure** event is generated. Note that by default, exposures is **:on** for new graphics contexts. See Section 5, Graphics Contexts, for further information.

*source* — Source **drawable**.

*gcontext* — The graphics context to use during the copy operation.

*source-x*, *source-y* — The x and y coordinates of the upper-left corner of the area in the *source* **drawable**. These coordinates are relative to the *source* **drawable** origin. Type is **int16**.

*width*, *height* — The width and height of the area being copied. These apply to both the *source* and *destination* areas. Type is **card16**.

*destination* — The destination **drawable**.

*destination-x*, *destination-y* — The x and y coordinates of the upper left corner of the area in the *destination* **drawable**. These coordinates are relative to the *destination* **drawable** origin. Type is **int16**.

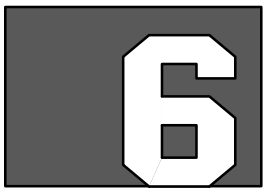
**copy-plane** *source gcontext plane source-x source-y width height destination destination-x destination-y* Function

Uses a single bit plane of the specified rectangular area of the *source* **drawable** along with the specified graphics context (*gcontext*) to modify the specified rectangle area of the *destination* **drawable**. The drawables specified by the *source* and *destination* arguments must have the same root but need not have the same depth.

Effectively, this operation forms a pixmap of the same depth as *destination* and with a size specified by the *source* area. It then uses the foreground and background from the graphics context (foreground where the bit-plane in *source* contains a one bit, background where the bit-plane in *source* contains a zero bit), and the equivalent of a **copy-area** operation is performed with all the same exposure semantics. This can also be thought of as using the specified region of the *source* bit-plane as a stipple with a fill-style of **:opaque-stippled** for filling a rectangular area of the *destination*.

*source* — The source **drawable**.

*gcontext* — The graphics context to use during the copy operation.



# GRAPHIC OPERATIONS

## Introduction

**6.1** Once connected to an X server, a client can use CLX functions to perform graphic operations on drawables.

This section describes CLX functions to:

- Operate on areas and planes
- Draw points
- Draw lines
- Draw rectangles
- Draw arcs
- Draw text

## Area and Plane Operations

**6.2 clear-area** clears an area or an entire window to the background. Since pixmaps do not have backgrounds, they cannot be filled by using the functions described in the following paragraphs. Instead, you should use **draw-rectangle**, which sets the pixmap to a known value. See paragraph 6.5, Drawing Rectangles, for information on **draw-rectangle**.

**clear-area** *window* &key (**:x** 0) (**:y** 0) **:width :height :exposures-p** Function

Draws a rectangular area in the specified *window* with the background pixel or pixmap of the *window*. The **:x** and **:y** coordinates are relative to the *window* origin, and specify the upper-left corner of the rectangular area that is to be cleared. A **nil** or zero value for **:height** or **:width** clears the remaining area (height – y or width – x). If the *window* has a defined background tile, the rectangle is tiled by using a plane-mask of all ones and a function of **:copy**. If the *window* has background **:none**, the contents of the *window* are not changed. In either case, if **:exposures-p** is non-**nil**, then one or more **:exposure** events are generated for regions of the rectangle that are either visible or are being retained in a backing store.

To clear the entire area in a specified *window*, use (**clear-area** *window*).

*window* — A **window**.

**:x**, **:y** — Upper-left corner of the area to be cleared. These coordinates are relative to the *window* origin. Type is **int16**.

**:width** — The width of the area to clear or **nil** to clear to the remaining width of the window. Type is **card16** or **null**.

**:height** — The height of the area to clear or **nil** to clear to the remaining height of the window. Type is **card16** or **null**.

**:exposures-p** — Specifies if **:exposure** events should be generated for the affected areas. Type **boolean**.