destination — The destination **gcontext**.

copy-gcontext-components *source destination* &rest *keys* Function

Copies the values of the specified attributes of the source graphics context to the destination graphics context. The source and destination graphics contexts must have the same root and depth.

source — The source **gcontext**.

destination — The destination **gcontext**.

keys — The remaining arguments are keywords, of type **gcontext-key**, which specify which attributes of the graphics context are to be copied.

Destroying Graphics Contexts

5.5 To destroy a graphics context, use **free-gcontext.**

free-gcontext *gcontext* Function

Deletes the association between the assigned resource ID and the specified graphics context, and then destroys the graphics context.

gcontext — A **gcontext**.

Graphics Context Cache

5.6 CLX provides a set of functions to control the automatic graphics context caching mechanism.

force-gcontext-changes *gcontext* Function

Forces any delayed changes to the specified graphics context to be sent out to the server. Note that **force-gcontext-changes** is called by all of the graphics functions.

gcontext — A **gcontext**.

with-gcontext *gcontext* &key **:arc-mode :background :cap-style :clip-mask :clip-ordering :clip-x :clip-y :dashes :dash-offset :exposures :fill-rule :fill-style :font :foreground :function :join-style :line-style :line-width :plane-mask :stipple :subwindow-mode :tile :ts-x :ts-y** &allow-other-keys &body *body* Macro

Changes the indicated graphics context components to the specified values only within the dynamic extent of the body. **with-gcontext** works on a per-process basis in a multiprocessing environment. The *body* is not surrounded by a **with-display** form. If there is no local cache for the specified graphics context, or if some of the component states are unknown, **with-gcontext** does the save and restore by creating a temporary graphics context and copying components to and from it using **copy-gcontext-components**.

gcontext — A **gcontext**.

:arc-mode, :background, **:cap-style**, **:clip-mask**, **:clip-ordering**, **:clip-x**, **:clip-y**, **:dashes**, **:dash-offset**, **:exposures**, **:fill-rule**, **:fill-style**, **:font**, **:foreground**, **:function**, **:join-style**, **:line-style**, **:line-width**, **:plane-mask**, **:stipple**, **:subwindow-mode**, **:tile**, **:ts-x**, **:ts-y** —These keyword arguments and associated values specify which graphics context components are to be changed. Any components not specified are left unmodified. See paragraph 5.2, Creating Graphics Contexts, for more information.

body — The body of code which will have access to the altered graphics context.

gcontext-ts-x *gcontext* Function

Returns:
ts-x — Type **int16**.

Returns the *ts-x* attribute of the specified graphics context.

The ts-x and ts-y attributes of a graphics context are the coordinates of the origin for tile pixmaps and the stipple.

gcontext — A **gcontext**.

gcontext-ts-y *gcontext* Function

Returns:
ts-y — Type **int16**.

Returns the *ts-y* attribute of the specified graphics context.

The ts-x and ts-y attributes of a graphics context are the coordinates of the origin for tile pixmaps and the stipple.

gcontext — A **gcontext**.

query-best-stipple *width height drawable* Function

Returns:
best-width, *best-height* — Type **card16**.

Returns the *best-width* and *best-height* for stipple pixmaps on the *drawable*.

The *drawable* indicates the screen and possibly the window class and depth. An **:input-only** window cannot be specified as the *drawable*. The size is returned as width and height values.

width, *height* — Specifies the width and height of the desired stipple pattern.

drawable — A **drawable**.

query-best-tile *width height drawable* Function

Returns:
best-width, *best-height* — Type **card16**.

Returns the *best-width* and *best-height* for tile pixmaps on the *drawable*.

The *drawable* indicates the screen and possibly the window class and depth. An **:input-only** window cannot be specified as the *drawable*. The size is returned as width and height values.

width, *height* — Specifies the width and height of the desired tile pattern.

drawable — A **drawable**.

Copying Graphics Contexts

5.3 CLX provides functions to copy some or all attribute values from one graphics context to another. These functions are generally more efficient than using **setf** to copy **gcontext** attributes individually.

copy-gcontext *source destination* Function

Copies all the values of the attributes of the source graphics context into the destination graphics context. The source and destination graphics contexts must have the same root and depth.

source — The source **gcontext**.

gcontext-subwindow-mode *gcontext* Function

Returns:
subwindow-mode — One of **:clip-by-children** or **:include-inferiors**.

Returns and (with **setf**) changes the subwindow-mode attribute of the specified graphics context.

The subwindow-mode attribute of a graphics context specifies whether subwindows obscure the contents of their parent window during a graphics operation. For a value of **:clip-by-children**, both source and destination windows are clipped by all viewable **:input-output** class children. This clipping is in addition to the clipping provided by the clip-mode attribute. For a value of **:include-inferiors**, neither the source nor destination window is clipped by its inferiors. This results in the inclusion of subwindow contents in the source and the drawing through of subwindow boundaries of the destination. The use of **:include-inferiors** on a window of one depth with mapped inferiors of differing depth is not illegal, but the semantics are not defined by the core protocol.

gcontext — A **gcontext**.

gcontext-tile *gcontext* Function

Returns:
tile — Type **pixmap**.

Returns the *tile* of the specified graphics context.

The tile attribute is a pixmap used to fill in areas for graphics operations. It is so named because copies of it are laid out side by side to fill the area.

The stipple and tile have the same origin. This origin point is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The stipple pixmap must have depth one and must have the same root as the graphics context. The tile pixmap must have the same root and depth as the graphics context. For stipple operations where the fill-style is **:stippled** (but not **:opaque-stippled**), the stipple pattern is tiled in a single plane and acts as an additional clip mask to be **and**ed with the clip-mask. Any size pixmap can be used for stipple or tile, although some sizes may be faster to use than others.

Specifying a pixmap for stipple or tile in a graphics context might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics operation, the change might or might not be reflected in the graphics context. If the pixmap is used both as the destination for a graphics operation and as a stipple or tile, the results are not defined.

Some displays have hardware support for tiling or stippling with patterns of specific sizes. Tiling and stippling operations that restrict themselves to those sizes may run much faster than such operations with arbitrary size patterns. CLX provides functions to determine the best size for stipple or tile (see **query-best-stipple** and **query-best-tile**).

gcontext — A **gcontext**.

The plane-mask attribute of a graphics context specifies which bit planes of the destination drawable are modified during a graphic operation. The plane-mask is a pixel value in which a 1 bit means that the corresponding bit plane will be modified and a 0 bit means that the corresponding bit plane will not be affected during a graphic operations. Thus, the actual result of a graphic operation depends on both the function and plane-mask attributes of the graphics context and is given by the following expression:

```
(logior (logand
          (boole function source destination)
          plane-mask)

        (logandc2
          destination
           plane-mask))
```

gcontext — A **gcontext**.

gcontext-plist *gcontext* Function

Returns:
gcontext-p — Type **list**.

Returns and (with **setf**) sets the property list for the specified *gcontext*. This function provides a hook where extensions can add data.

gcontext — A **gcontext**.

gcontext-stipple *gcontext* Function

Returns:
stipple — Type **pixmap**.

Returns the *stipple* of the specified graphics context.

The stipple attribute of a graphics context is a bitmap used to prevent certain pixels in the destination of graphics operations from being affected by tiling.

The stipple and tile have the same origin. This origin point is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The stipple pixmap must have depth one and must have the same root as the graphics context. The tile pixmap must have the same root and depth as the graphics context. For stipple operations where the fill-style is **:stippled** (but not **:opaque-stippled**), the stipple pattern is tiled in a single plane and acts as an additional clip mask to be **and**ed with the clip-mask. Any size pixmap can be used for stipple or tile, although some sizes may be faster to use than others.

Specifying a pixmap for stipple or tile in a graphics context might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics operation, the change might or might not be reflected in the graphics context. If the pixmap is used both as the destination for a graphics operation and as a stipple or tile, the results are not defined.

Some displays have hardware support for tiling or stippling with patterns of specific sizes. Tiling and stippling operations that restrict themselves to those sizes may run much faster than such operations with arbitrary size patterns. CLX provides functions to determine the best size for stipple or tile (see **query-best-stipple** and **query-best-tile**).

gcontext — A **gcontext**.

Wide lines are drawn centered on the path described by the graphics operation. Unless otherwise specified by the join-style or cap-style, the bounding box of a wide line with end points [x1, y1], [x2, y2], and width w is a rectangle with vertices at the following real coordinates:

[x1 – (w*sin*/2), y1 + (w*cos*/2)], [x1+ (w*sin*/2), y1 – (w*cos*/2)],
[x2 – (w*sin*/2), y2 + (w*cos*/2)], [x2 + (w*sin*/2), y2 – (w*cos*/2)]

where *sin* is the sine of the angle of the line and *cos* is the cosine of the angle of the line. A pixel is part of the line and, hence, is drawn if the center of the pixel is fully inside the bounding box (which is viewed as having infinitely thin edges). If the center of the pixel is exactly on the bounding box, it is part of the line if, and only if, the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if, and only if, the interior is immediately below (y increasing direction).

Thin lines (zero line-width) are always one pixel wide lines drawn using an unspecified, device dependent algorithm. There are only two constraints on this algorithm.

1. If a line is drawn unclipped from [x1,y1] to [x2,y2] and if another line is drawn unclipped from [x1+dx,y1+dy] to [x2+dx,y2+dy], a point [x,y] is touched by drawing the first line if, and only if ,the point [x+dx,y+dy] is touched by drawing the second line.

2. The effective set of points comprising a line cannot be affected by clipping. That is, a point is touched in a clipped line if, and only if, the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

A wide line drawn from [x1,y1] to [x2,y2] always draws the same pixels as a wide line drawn from [x2,y2] to [x1,y1], not counting cap-style and join-style. Implementors are encouraged to make this property true for thin lines, but it is not required. A line-width of zero may differ from a line-width of one in which pixels are drawn. This permits the use of many manufacturer's line drawing hardware, which may run much faster than the more precisely specified wide lines.

In general, drawing a thin line is faster than drawing a wide line of width one. However, because of their different drawing algorithms, thin lines may not mix well, aesthetically speaking, with wide lines. If it is desirable to obtain precise and uniform results across all displays, a client should always use a line-width of one, rather than a line-width of zero.

gcontext — A **gcontext**.

gcontext-p *gcontext* Function

Returns:
gcontext — Type **boolean**.

Returns non-**nil** if the argument is a graphics context and **nil** otherwise.

gcontext-plane-mask *gcontext* Function

Returns:
plane-mask — Type **card32**.

Returns the *plane-mask* of the specified graphics context.

The join-style attribute of a graphics context defines how the segment intersections are drawn for wide polylines. The possible values and their interpretations are as follows:

Join-Style	Interpretation
:bevel	Uses **:butt** end point styles with the triangular notch filled.
:miter	The outer edges of two lines extend to meet at an angle.
:round	A circular arc with diameter equal to the line-width, centered on the join point.

When the end points of a polyline segment are identical, the effect is as if the segment was removed from the polyline. When a polyline is a single point, the effect is the same as when the cap-style is applied at both end points.

gcontext — A **gcontext**.

gcontext-line-style *gcontext* Function

Returns:
line-style — One of **:dash**, **:double-dash**, or **:solid**.

Returns and (with **setf**) changes the line-style attribute of the specified graphics context.

The line-style attribute of a graphics context specifies how (which sections of) lines are drawn for a path in graphics operations. The possible values and their meanings are as follows:

Line-Style	Meaning
:solid	The full path is drawn.
:double-dash	The full path is drawn, but the even dashes are filled differently than the odd dashes. The **:butt** style is used where even and odd dashes meet (see paragraph 5.4.7, Fill-Rule and Fill-Style).
:on-off-dash	Only the even dashes are drawn, with cap-style applied to all internal ends of the individual dashes, except **:not-last** is treated as **:butt**.

gcontext — A **gcontext**.

gcontext-line-width *gcontext* Function

Returns:
line-width — Type **card16**.

Returns the *line-width* of the specified graphics context.

The line-width is measured in pixels and can be greater than or equal to one (wide line) or can be the special value zero (thin line).

Symbol	Result
boole-1	*S*
boole-2	*D*
boole-andc1	(logandc1 *S D*)
boole-andc2	(logandc2 *S D*)
boole-and	(logand *S D*)
boole-c1	(lognot *S*)
boole-c2	(lognot *D*)
boole-clr	0
boole-eqv	(logeqv *S D*)
boole-ior	(logior *S D*)
boole-nand	(lognand *S D*)
boole-nor	(lognor *S D*)
boole-orc1	(logorc1 *S D*)
boole-orc2	(logorc2 *S D*)
boole-set	1
boole-xor	(logxor *S D*)

gcontext — A **gcontext**.

gcontext-id *gcontext* Function

Returns:
id — Type **resource-id**.

Returns the unique ID that has been assigned to the specified graphics context.

gcontext — A **gcontext**.

gcontext-join-style *gcontext* Function

Returns:
join-style — One of **:bevel**, **:miter**, or **:round**.

Returns and (with **setf**) changes the join-style attribute of the specified graphics context.

gcontext — A **gcontext**.

gcontext-font *gcontext* &optional *metrics-p* Function

Returns:
font — Type **font** or **null**.

Returns and (with **setf**) changes the *font* attribute of the specified graphics context. If the stored font is known, it is returned. If it is not known and the *metrics-p* argument is **nil**, then **nil** is returned. If the font is not known and *metrics-p* is true, then a pseudo-font is constructed and returned. For a constructed pseudo-font, full metric and property information can be obtained, but it does not have a name or a resource ID, and attempts to use it where a resource ID is required results in an invalid-font error.

The font attribute in a graphics context defines the default text font used in text drawing operations. When setting the value of the font attribute, either a **font** object or a font name can be used. If a font name is passed, **open-font** is call automatically to get the **font** object.

gcontext — A **gcontext**.

metrics-p — Specifies whether a pseudo-font is returned when the real font stored in the graphics context is not known. The default is **nil**, which means do not return a pseudo-font.

gcontext-foreground *gcontext* Function

Returns:
foreground — Type **card32**.

Returns and (with **setf**) changes the foreground attribute of the specified graphics context.

The foreground attribute of a graphics context specifies the pixel value drawn for set bits in a bitmap and for bits set by a graphics operation.

gcontext — A **gcontext**.

gcontext-function *gcontext* Function

Returns:
function — Type **boole-constant**.

Returns the *function* of the specified graphics context.

In all graphic operations, given a source pixel and a corresponding destination pixel, the resulting pixel drawn is computed bitwise on the bits of the source and destination pixels. That is, a logical operation is used to combine each bit plane of corresponding source and destination pixels. The graphics context function attribute specifies the logical operation used via one of the 16 operation codes defined by Common Lisp for the **boole** function.

The following table shows each of the logical operation codes that can be given by the function attribute. For each operation code, its result is shown as a logical function of a source pixel *S* and a destination pixel *D*.

The **:winding** rule defines a point to be inside if any infinite ray starting at the point crosses an unequal number of clockwise and counterclockwise directed border segments. A clockwise directed border segment crosses the ray from left to right as observed from the point. A counterclockwise segment crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because you can simply choose a different ray that is not coincident with a segment.

For both **:even-odd** and **:winding**, a point is infinitely small, and the border is an infinitely thin line. A pixel is inside if the center point of the pixel is inside, and the center point is not on the border. If the center point is on the border, the pixel is inside if, and only if, the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if, and only if, the polygon interior is immediately below (y increasing direction).

gcontext — A **gcontext**.

gcontext-fill-style *gcontext* Function

Returns:
fill-style — One of **:opaque-stippled**, **:solid**, **:stippled**, or :**tiled**.

Returns and (with **setf**) changes the fill-style attribute of the specified graphics context.

The fill-style attribute of a graphics context defines the contents of the source for line, text, and fill graphics operations. It determines whether the source image is drawn with a solid color, a tile, or a stippled tile. The possible values and their meanings are as follows:

Fill-Style	Meaning
:opaque-stippled	Filled with a tile with the same width and height as stipple, but with the background value used everywhere stipple has a zero and the foreground pixel value used everywhere stipple has a one.
:solid	Filled with the foreground pixel value.
:stippled	Filled with the foreground pixel value masked by stipple.
:tiled	Filled with tile.

When drawing lines with line-style **:double-dash**, the filling of the odd dashes are controlled by the fill-style in the following manner:

Fill-Style	Effect
:opaque-stippled	Same as for even dashes.
:solid	Filled with the background pixel value.
:stippled	Filled with the background pixel value masked by stipple.
:tiled	Filled the same as the even dashes.

The dashes attribute in a graphics context specifies the pattern that is used for graphics operations which use the dashed line styles. It is a non-**nil** sequence with each element representing the length of a single dash or space. The initial and alternating elements of the dashes are the even dashes, while the others are the odd dashes. An odd length sequence is equivalent to the same sequence concatenated with itself to produce an even length sequence. All of the elements of a dashes sequence must be non-zero.

Specifying a single integer value, *N*, for the dashes attribute is an abbreviated way of specifying a two element sequence with both elements equal to the specified value [*N*, *N*].

The unit of measure for dashes is the same as in the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but server implementations are only required to match this ideal for horizontal and vertical lines.

gcontext — A **gcontext**.

gcontext-display *gcontext* Function

Returns:
display — Type **display**.

Returns the **display** object associated with the specified *gcontext*.

gcontext — A **gcontext**.

gcontext-equal *gcontext-1 gcontext-2* Function

Returns:
equal-p — Type **boolean**.

Returns true if the two arguments refer to the same server resource, and **nil** if they do not.

gcontext-1, *gcontext-2* — A **gcontext**.

gcontext-exposures *gcontext* Function

Returns:
exposures — Either **:off** or **:on**.

Returns and (with **setf**) changes the exposures attribute of the specified graphics context.

The exposures attribute in a graphics context controls the generation of **:graphics-exposure** events for calls to the **copy-area** and **copy-plane** functions. If **:on**, **:graphics-exposure** events will be reported when calling the **copy-area** and **copy-plane** functions with this graphics context. Otherwise, if **:off**, the events will not be reported.

gcontext — A **gcontext**.

gcontext-fill-rule *gcontext* Function

Returns:
fill-rule — Either **:even-odd** or **:winding**.

Returns and (with **setf**) changes the fill-rule attribute of the specified graphics context.

The fill-rule attribute in a graphics context specifies the rule used to determine the interior of a filled area. It can be specified as either **:even-odd** or **:winding**.

The **:even-odd** rule defines a point to be inside if any infinite ray starting at the point crosses the border an odd number of times. Tangencies do not count as a crossing.

If known by the client, the ordering of clip-mask rectangles can be specified to provide faster operation by the server. A value of **:unsorted** means the rectangles are in arbitrary order. A value of **:y-sorted** means that the rectangles are non-decreasing in their Y origin. A **:yx-sorted** value is like **:y-sorted** with the additional constraint that all rectangles with an equal Y origin are non-decreasing in their X origin. A **:yx-banded** value additionally constrains **:yx-sorted** by requiring that, for every possible Y scan line, all rectangles that include that scan line have an identical Y origins and Y extents. If incorrect ordering is specified, the X server may generate an error, but it is not required to do so. If no error is generated, the results of the graphics operations are undefined.

gcontext — A **gcontext**.

ordering — One of **:unsorted**, **:y-sorted**, **:yx-banded**, **:yx-sorted**, or **nil**.

gcontext-clip-x *gcontext* Function

Returns:
clip-x — Type **int16**.

Returns and (with **setf**) changes the clip-x attribute of the specified graphics context.

The clip-x and clip-y attributes specify the origin for the clip-mask, whether it is a pixmap or a sequence of rectangles. These coordinates are interpreted relative to the origin of whatever destination drawable is specified in a graphics operation.

gcontext — A **gcontext**.

gcontext-clip-y *gcontext* Function

Returns:
clip-y — Type **int16**.

Returns and (with **setf**) changes the clip-y attribute of the specified graphics context.

The clip-x and clip-y attributes specify the origin for the clip-mask, whether it is a pixmap or a sequence of rectangles. These coordinates are interpreted relative to the origin of whatever destination drawable is specified in a graphics operation.

gcontext — A **gcontext**.

gcontext-dash-offset *gcontext* Function

Returns:
dash-offset — Type **card16**.

Returns and (with **setf**) changes the dash-offset attribute of the specified graphics context.

The dash-offset attribute of a graphics context defines the phase of the pattern contained in the dashes attribute. This phase specifies how many elements (pixels) into the path the pattern should actually begin in any single graphics operation. Dashing is continuous through path elements combined with a join-style, but is reset to the dash-offset each time a cap-style is applied at a line end point.

gcontext — A **gcontext**.

gcontext-dashes *gcontext* Function

Returns:
dashes — Type **sequence** or **card8**.

Returns and (with **setf**) changes the dashes attribute of the specified graphics context. The sequence must be non-empty and the elements must be non-zero **card8** values.

Cap-Style	Line-Width	Effect
:butt	thin	Device dependent, but the desired effect is that a single pixel is drawn.
:butt	wide	Nothing is drawn.
:not-last	thin	Device dependent, but the desired effect is that nothing is drawn.
:projecting	thin	Same as **:butt** with thin line-width.
:projecting	wide	The closed path is a square, aligned with the coordinate axes, centered at the end point, with sides equal to the line-width.
:round	wide	The closed path is a circle, centered at the end point, with diameter equal to the line-width.
:round	thin	Same as **:butt** with thin line-width.

gcontext — A **gcontext**.

gcontext-clip-mask *gcontext* &optional *ordering* Function

Returns and (with **setf**) changes the clip-mask attribute of the graphics context.

When changing the clip-mask attribute, the new clip-mask can be specified as a pixmap or a **rect-seq** or as the values **:none** or **nil**. The ordering argument can be specified only with **setf** when the new clip-mask is a **rect-seq**.

The clip-mask attribute of a graphics context affects all graphics operations and is used to restrict output to the destination drawable. The clip-mask does not clip the source of a graphics operation. A value of **:none** for clip-mask indicates that no clipping is to be done.

If a pixmap is specified as the clip-mask, it must have depth one and the same root as the specified graphics context. Pixels where the clip-mask has a one bit are drawn. Pixels outside the area covered by the clip-mask or where the clip-mask has a zero bit are not drawn.

If a sequence of rectangles is specified as the clip-mask, the output is clipped to remain contained within the rectangles. The rectangles should be non-intersecting, or the results of graphics operations will be undefined. The rectangle coordinates are interpreted relative to the clip origin. Note that the sequence of rectangles can be empty, which effectively disables output. This is the opposite of setting the clip-mask to **:none**.

gcontext — A **gcontext**.

gcontext-background *gcontext* Function

Returns:
background — Type **card32**.

Returns and (with **setf**) changes the background attribute of the specified graphics context.

The background attribute specifies the pixel value drawn for pixels that are not set in a bitmap and for pixels that are cleared by a graphics operation, such as the gaps in dashed lines.

gcontext — A **gcontext**.

gcontext-cache-p *gcontext* Function

Returns:
cache-p — Type **boolean**.

Returns and (with **setf**) changes the local cache mode for the *gcontext*. If true, the state of the *gcontext* is cached by CLX and changes to its attributes have no effect unless the new value differs from its cached value.

gcontext — A **gcontext**.

gcontext-cap-style *gcontext* Function

Returns:
cap-style — One of **:butt**, **:not-last**, **:projecting**, or **:round**.

Returns and (with **setf**) changes the cap-style attribute of the specified graphics context.

The cap-style attribute of a graphics context defines how the end points of a path are drawn. The possible values and their interpretations are as follows:

Cap-Style	Interpretation
:butt	Square at the end point (perpendicular to the slope of the line) with no projection beyond.
:not-last	Equivalent to **:butt**, except that for a line-width of zero or one the final end point is not drawn.
:projecting	Square at the end, but the path continues beyond the end point for a distance equal to half the line-width. This is equivalent to **:butt** for line-width zero or one.
:round	A circular arc with the radius equal to 1/2 of the line-width, centered on the end point. This is equivalent to **:butt** for line-width zero or one.

The following table describes what happens when the end points of a line are identical. The effect depends on both the cap style and line width.

All of the graphics context components are set to the values that are specified by the keyword arguments, except that a value of **nil** causes the default value to be used. These default values are as 6.follows:

Component	Default Value
arc-mode	**:pie-slice**
background	1
cap-style	**:butt**
clip-mask	**:none**
clip-ordering	**:unsorted**
clip-x	0
clip-y	0
dash-offset	0
dashes	4 (that is, the list '(4, 4))
exposures	**:on**
fill-rule	**:even-odd**
fill-style	**:solid**
font	server dependent
foreground	0
function	**boole-1**
join-style	**:miter**
line-style	**:solid**
line-width	0
plane-mask	A bit mask of all ones
stipple	Pixmap of unspecified size filled with ones
subwindow-mode	**:clip-by-children**
tile	Pixmap of an unspecified size filled with the foreground pixel (that is, the client-specified pixel if any, or else 0)
ts-x	0
ts-y	0

Note that foreground and background do not default to any values that are likely to be useful on a color display. Since specifying a **nil** value means use the default, this implies for clip-mask that an empty rectangle sequence cannot be specified as an empty list; **:none** must be used instead. Specifying a **stringable** for font causes an implicit **open-font** call to occur.

Graphics Context Attributes

5.3 The following paragraphs describe the CLX functions used to return or change the attributes of a **gcontext**. Functions that return the contents of a **gcontext** return **nil** if the last value stored is unknown (for example, if the **gcontext** was not cached or if the **gcontext** was not created by the inquiring client).

gcontext-arc-mode *gcontext* Function

Returns:

arc-mode — Either **:chord** or **:pie-slice**.

Returns and (with **setf**) changes the arc-mode attribute of the specified graphics context.

The arc-mode attribute of a graphics context controls the kind of filling, if any, to be done by the **draw-arcs** function. A value of **:chord** specifies that arcs are filled inward to the chord between the end points of the arc. **:pie-slice** specifies that arcs are filled inward to the center point of the arc, creating a pie slice effect.

Creating Graphics Contexts

5.2 To create a graphics context, use **create-gcontext.**

create-gcontext &key **:arc-mode :background** (**:cache-p** t) **:cap-style** **:clip-mask :clip-ordering :clip-x :clip-y :dash-offset :dashes** **:drawable :exposures :fill-rule :fill-style :font :foreground** **:function :join-style :line-style :line-width :plane-mask :stipple** **:subwindow-mode :tile :ts-x :ts-y** Function

Returns:

gcontext — Type **gcontext**.

Creates, initializes, and returns a graphics context (**gcontext**). The graphics context can only be used with destination drawables having the same root and depth as the specified **:drawable**. If **:cache-p** is non-**nil**, the graphics context state is cached locally, and changing a component has no effect unless the new value differs from the cached value. Changes to a graphics context (**setf** and **with-gcontext**) are always deferred regardless of the cache mode and sent to the server only when required by a local operation or by an explicit call to **force-gcontext-changes**.

:cache-p — Specifies if this graphics context should be cached locally by CLX. If **nil** then the state is not cached, otherwise a local cache is kept.

:drawable — The **drawable** whose root and depth are to be associated with this graphics context. This is a required keyword argument.

:arc-mode, **:background**, **:cap-style**, **:clip-mask**, **:clip-ordering**, **:clip-x**, **:clip-y**, **:dash-offset**, **:dashes**, **:exposures**, **:fill-rule**, **:fill-style**, **:font**, **:foreground**, **:function**, **:join-style**, **:line-style**, **:line-width**, **:plane-mask**, **:stipple**, **:subwindow-mode**, **:tile**, **:ts-x**, **:ts-y** — Initial attribute values for the graphics context.

GRAPHICS CONTEXTS

Introduction

5.1 Clients of the X Window System specify the visual attributes of graphical output primitives by using *graphics contexts*. A graphics context is a set of graphical attribute values such as foreground color, font, line style, and so forth. Like a window, a graphics context is another kind of X server resource which is created and maintained at the request of a client program. The client program, which may use several different graphics contexts at different times, is responsible for specifying a graphics context to use with each graphical output function.

CLX represents a graphics context by an object of type **gcontext** and defines functions to create, modify, and manipulate **gcontext** objects. By default, CLX also records the contents of graphics contexts in a cache associated with each display. This local caching of graphics contexts has two important advantages:

1. Communication efficiency — Changes to attribute values in a **gcontext** are first made only in the local cache. Just before a **gcontext** is actually used, CLX automatically sends any changes to the X server, batching all changes into a single request.

2. Inquiring **gcontext** contents — Accessor functions can be used to return the value of any individual **gcontext** component by reading the copy of the **gcontext** from the cache. This kind of inquiry is not supported by the basic X protocol. There is no way for a client program to request an X server to return the contents of a **gcontext**.

Caching graphics contexts can result in a synchronization problem if more than one client program modifies a graphics context. However, this problem is unusual. Sharing a graphics context among several clients, while possible, is not expected to be useful and is not very easy to do. At any rate, a client program can choose to not cache a **gcontext** when it is created.

Each client program must determine its own policy for creating and using graphics contexts. Depending on the display hardware and the server implementation, creating a new graphics context can be more or less expensive than modifying an existing one. In general, some amount of graphics context information can be cached in the display hardware, in which case modifying the hardware cache is faster than replacing it. Typical display hardware can cache only a small number of graphics contexts. Graphics output is fastest when only a few graphics contexts are used without heavy modifications.

This section explains the CLX functions used to:

- Create a graphics context
- Return the contents of a graphics context
- Change the contents of a graphics context
- Copy a graphics context
- Free a graphics context