Symbolics X Window System User's Guide

The Genera X Client

Configuring the Remote Screen Facility for the X Client

Before you configure the remote screen facility, restore the distribution tape as follows:

1. Place the Symbolic's X Window System tape in the tape drive

2. Specify `Restore Distribution :menu Yes`

3. Click on `Initialize Restoration`

4. Click on `Perform Restoration`

Next you need to load the remote screen facility, as follows:

1. Be sure that you have the RPC, Embedding-Support, and IP-TCP systems loaded in the world you are using. You can check by using the Show Herald command. If any of these systems are not loaded, load them individually with Load System.

2. Load the system X-Remote-Screen.

Configuring the remote screen facility for the Symbolics X window system consists of these steps:

- Installing Genera fonts for your X server.

- Providing X server access.

Installing Genera Fonts for the X Client

The Genera X Client can make use of either the Genera X fonts or the the standard 75dpi X fonts. The easiest thing to do is to use the standard 75dpi X fonts provided with the X Window server you are using. However, if you want your remote X consoles to use the same fonts as local consoles, use the Genera fonts.

The next sections tell you how to install the Genera fonts for the X Window server you are using. Instructions are given here for users of the Sun Open Window system and the M.I.T. X server.

Installing Genera Fonts for Open Windows

No documentation available for section Installing Genera Fonts for Open Windows.

Installing Genera Fonts for the M.I.T. X Server

For the M.I.T. X server, Genera fonts should be compiled by the bdftosnf program from the X distribution and installed in the directory /usr/lib/X11/fonts/genera.

In this example, we assume that you are using the M.I.T. X server, that the UNIX software tape was restored to /usr/share/local/ivory, that the bdftosnf program is installed in the directory /usr/bin/X11, that the compiled fonts are to be installed in the directory /usr/lib/X11/fonts/genera, and that you are using the csh shell. The example compiles each BDF file in /usr/share/local/ivory/fonts into the corresponding SNF file in /usr/lib/X11/fonts/genera.

```
% cd /usr/share/local/ivory/fonts
% mkdir /usr/lib/X11/fonts/genera
% foreach font ( *.bdf )
? /usr/bin/X11/bdftosnf $font > `echo $font \
  | sed -e 's/bdf$/snf/' \
  | awk '{print "/usr/lib/X11/fonts/genera/" $1}'`
? end
% cd /usr/lib/X11/fonts/genera
% /usr/bin/X11/mkfontdir
```

Once you have compiled the fonts, you must add the Genera fonts directory to your X server's font path.

No documentation available for section Example of Setting the Font Path for the M.I.T. X Server.

Providing X Server Access to the Symbolics Machine

Because Genera needs access to the X display for its console screen, the X server needs to be told to allow access from the Symbolics computer. Since X server access control only lasts for the duration of an X session, you will have to do this each time you log into your machine.

In the M.I.T. X Window System distribution, access can be enabled with the xhost UNIX program. Use the xhost UNIX program to tell the X server to allow the Symbolics computer to open connections to the display. See the UNIX manual page for the xhost UNIX program for more information.

If you are using OpenWindows, use the newshost program.

Connecting to a Genera Application

The X Window System is a network protocol, and requires a *rendezvous* between a client and server before communications can take place. In the X Window System, the client is the application seeking to present a user interface, and the server is a virtual console capable of doing so. In the case of the Symbolics UX, Genera running on the coprocessor is the X client, and the host workstation (or any other workstation on the network) is the X server. The Genera X client must be specifically requested to connect to a given X server.

When using the Genera X client from a non-UNIX workstation such as an X terminal or a Symbolics workstation, you must independently connect to the Genera Command Processor and use the Start X Screen command. You can also use TELNET from a non-UNIX workstation. When using the Genera X client from a UNIX workstation, use the `genera` UNIX program to establish and supervise the X connection. The `genera` program uses a simple RPC protocol to contact Genera and invoke the Start X Screen command. See the section "`genera` UNIX Program".

The Genera X client supports multiple user interfaces simultaneously, possibly displayed on different consoles. This does not, however, make Genera a multiuser operating system. The Genera operating system provides a single address space, or environment, for all applications running within it. There are no mechanisms to isolate multiple applications (or multiple copies of a single application) from each other, so applications intended for use by multiple users must carefully define which objects are shared among users and which are strictly per-user. **dw:define-program-framework** encourages writing programs that can be used by several users independently, by providing a per-invocation repository for application objects.

Start X Screen Command

Start X Screen *host keywords*

Connects to an X server to present the Genera user interface. Start X Screen takes a number of optional arguments, principally one to specify the activity which is to run in the new screen. If no activity is specified, a generic Genera console and user interface is presented, and various activities may be selected within it using the usual mechanisms.

The following arguments specify where the user interface is to be displayed:

host	The name of the host on which to display the user interface. It must be running an X server and provide X-WINDOW-SYSTEM service.
:Display	The number of the display (X terminology).
:Screen	The number of the screen (X terminology).

The following keyword arguments specify the Genera application in the new screen:

:Activity	The name of the Genera activity to use.
:Program	The name of the remote program to use.
:Protocol	The network protocol to use for the connection.

There are a number of arguments that specify visual characteristics of the new screen. The default is to create a screen that covers most of the X server's console, with a who line (also called a status line).

:Geometry	An X geometry specification (for example, `785x800+100+100`)
:Who Line	A Boolean value indicating whether to append a status line to the screen.
:Foreground	The foreground color.
:Background	The background color.
:Border Width	The width of the screen border, in pixels.
:Border Color	The color of the border.
:Compatible Color	Whether or not the screen supports compatible color.
:Initial State	:normal or :iconic, indicating whether the new window should be initially iconified or not.

Start X Screen normally attempts to use the resources (screens, bitmaps, dynamic window histories) of previously created screens that have been disconnected, by Halt X Screen, warm booting Genera, or by a failure of the network connection. This behavior may be disabled using the **:reuse** argument.

:Reuse	A Boolean value indicating whether to reuse an old screen or force a new one to be created.

The following example presents the user interface of a Genera main screen on the primary console of host ENIAC, assuming that ENIAC is running an X server. The window will appear with an image of a Dynamic Lisp Listener inside it; other Genera activities may be selected and will appear within the window, as though it were the console of a Symbolics workstation.

```
Start X Screen (host) ENIAC
```

The following example presents the user interface of the Genera Zmacs editor on the primary console of host ENIAC. The window will appear with Zmacs' default width of 785 pixels, and without a Genera status line.

```
Start X Screen (host) ENIAC :Activity Zmacs
```

Halt X Screen Command

Halt X Screen *screen*

Disconnects an established X connection, destroying the corresponding X window. The *screen* argument defaults to the console from which the Halt X Screen command was executed, if that is an X screen. Otherwise, the *screen* argument is hard to type: type `help` and use the mouse to indicate which of the possibilities you intended.

Keyboard Support in the Genera X Client

Genera was designed for use with a keyboard that includes a rich selection of modifier keys (CONTROL, META, SUPER, and so on) and a number of special-function keys (HELP, COMPLETE, ABORT, SUSPEND, RESUME, and so on). This keyboard is exemplified by the one shipped with Symbolics workstations. With the advent of remote console support, such as the X Window System, Genera is increasingly used from consoles with other, widely varying, types of keyboards.

Genera accommodates various other keyboards by translating keystrokes from the physical keyboard into its own abstract set of keystrokes. The Genera X client requires that the X server support the following keystrokes (which might be synthesized by the X server if they are not present on the physical keyboard): full alphanumeric keys, control, meta, and alt modifiers, separate DELETE and BACKSPACE keys, separate RETURN and LINEFEED keys, and twelve general-purpose function keys. Every Genera keystroke may be specified using any keyboard that meets these minimum requirements; the general-purpose function keys are translated into the following Genera keystrokes:

Function Key	*Value without Shift Key*	*Value with Shift Key*
F1	Select	Square
F2	Network	Circle
F3	Function	Triangle
F4	Suspend	Mode Lock
F5	Resume	
F6	Abort	
F7	Super (modifier)	
F8	Hyper (modifier)	
F9	Scroll	Page
F10	Clear Input	Refresh
F11	Complete	End
F12	Help	

Some keyboards have more convenient locations for some of these keys. For example, many keyboards have a HELP key somewhere. The Genera X client recognizes certain popular keyboards, and customizes the keyboard layout for them. The customization is done only by making copies of the keystrokes on the function keys, never by moving them. So, on a keyboard with a HELP key, Genera's Help gesture may be invoked by pressing either HELP or F12. The "Show Keyboard Layout Command" will display the actual keyboard layout on the screen.

The Symbolics X Client software recognizes the Sun Type-3 and Type-4 keyboards, and will customize the keyboard layout for them. See the section "Sun Keyboards".

Utilities provided with the X Window System running on your X Server (for example, a UNIX machine or an NCD X Terminal) may be used to customize the keyboard layout. The Show X Keyboard Mapping command provides detailed information about the translation of X keycodes to Genera keystrokes.

See the section "Show X Keyboard Mapping Command". Note that the recognition algorithm tries to accommodate a certain amount of customization by the X server, but heavily customized keyboards may cause the keyboard not to be recognized, in which case only the standard function key mappings will be available.

Note: The customized keyboard layouts used by the Genera X client are not used in the Symbolics UX cold load stream, only the standard function key bindings are available. See the section "ASCII Keyboard Mappings".

The Symbolics keyboard control facility can be used by the Genera X Client. To access it, use the Select Activity command from a Lisp Listener, specifying the argument `Keyboard Control`.

Some keyboards contain interesting keys that aren't in the standard Genera character set. These keys are translated into a special character set called the Keyboard character set, so that commands may be attached to them. See the section "The Keyboard Character Set".

Sun Keyboards

The Symbolics X Client software recognizes the Sun Type 3 and Type 4 keyboards, and will customize the keyboard mapping for them as shown in 107 and 108 .

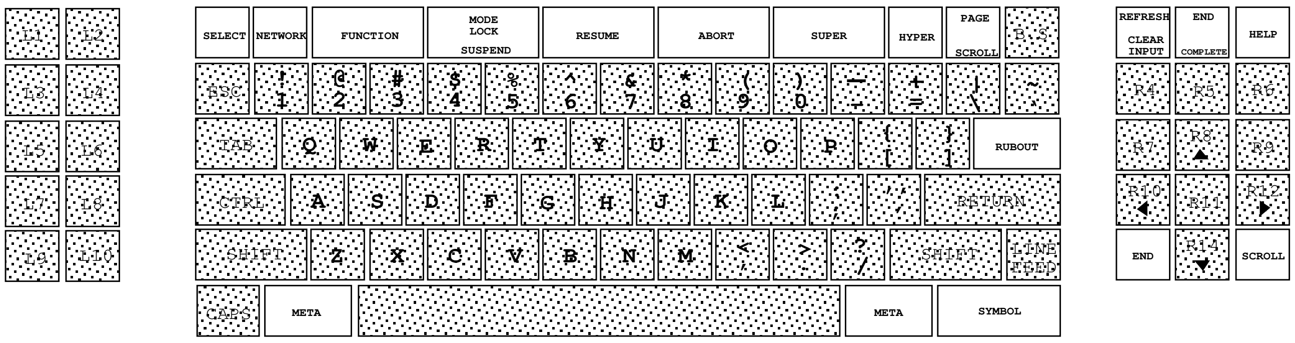

Figure 121. Genera Interpretation of the Sun Type 3 Keyboard

Note: On a UX with a Type 4 keyboard, Genera uses the `F11` key as the `COMPLETE` key. However, the cold load window uses the `R2` key for `COMPLETE`.

Default NCD N-101 to Symbolics Keyboard Mappings

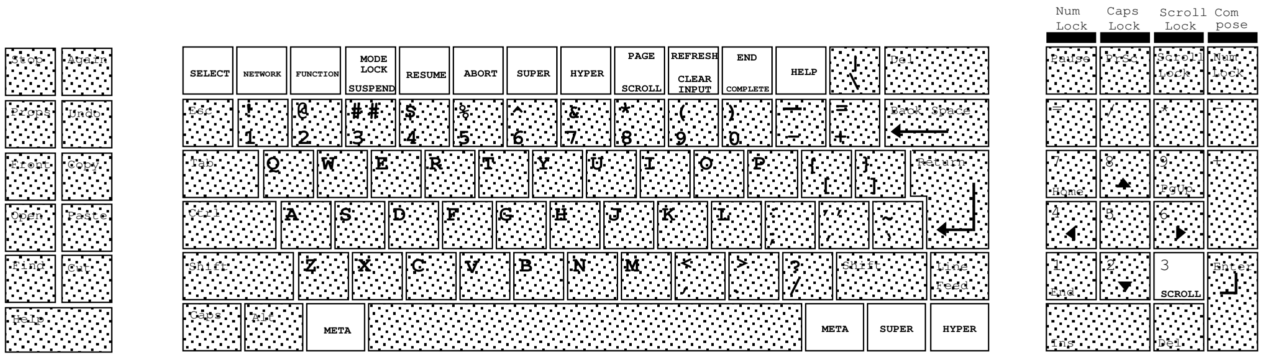

Figure 122. Genera Interpretation of the Sun Type 4 Keyboard

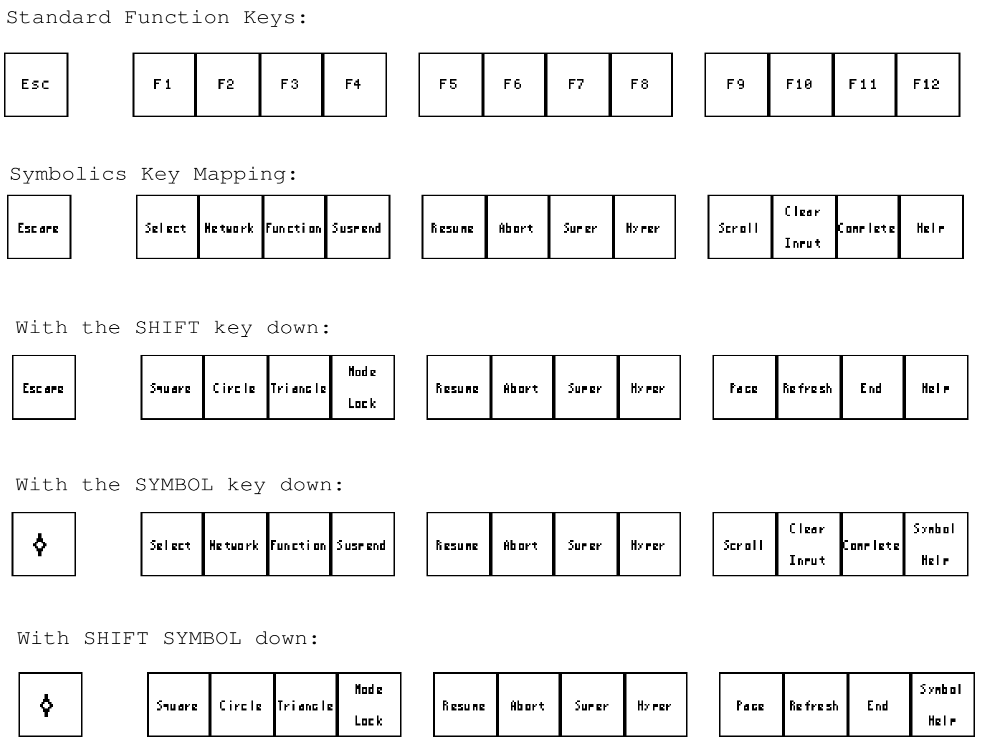

Figure 123. Function keys

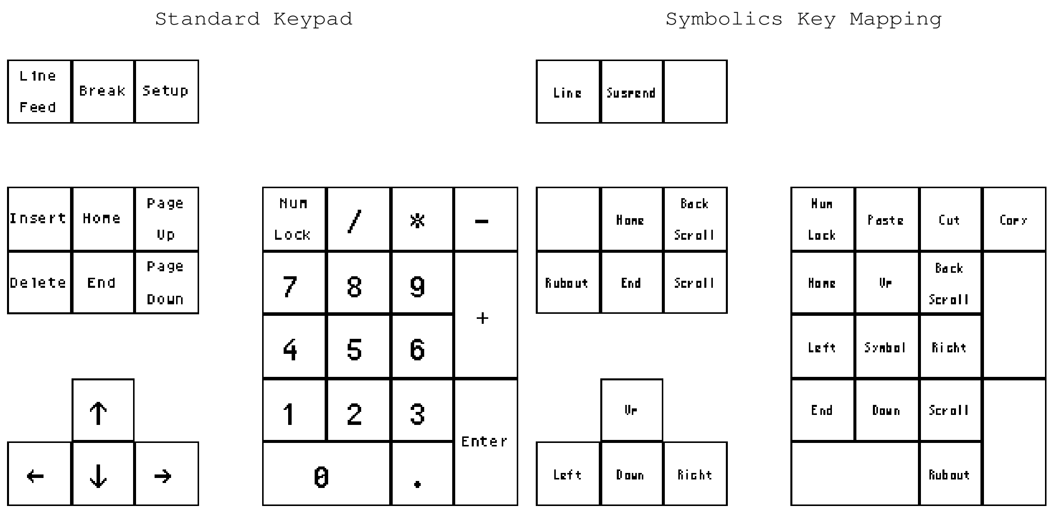

Figure 124. The Keypad

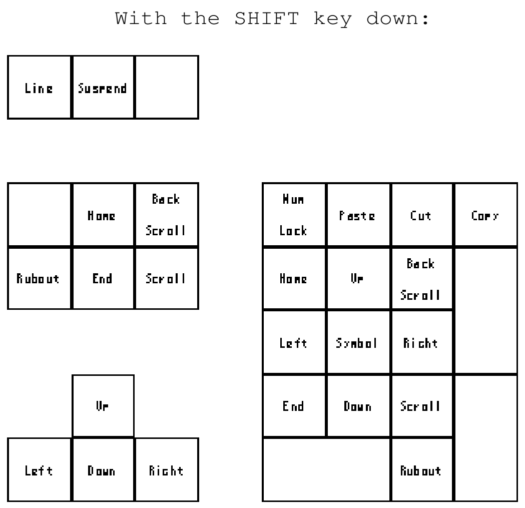

Figure 125. The Keypad with the Shift Key as Modifier

Show Keyboard Layout Command

Show Keyboard Layout *keyboard-layout keywords*

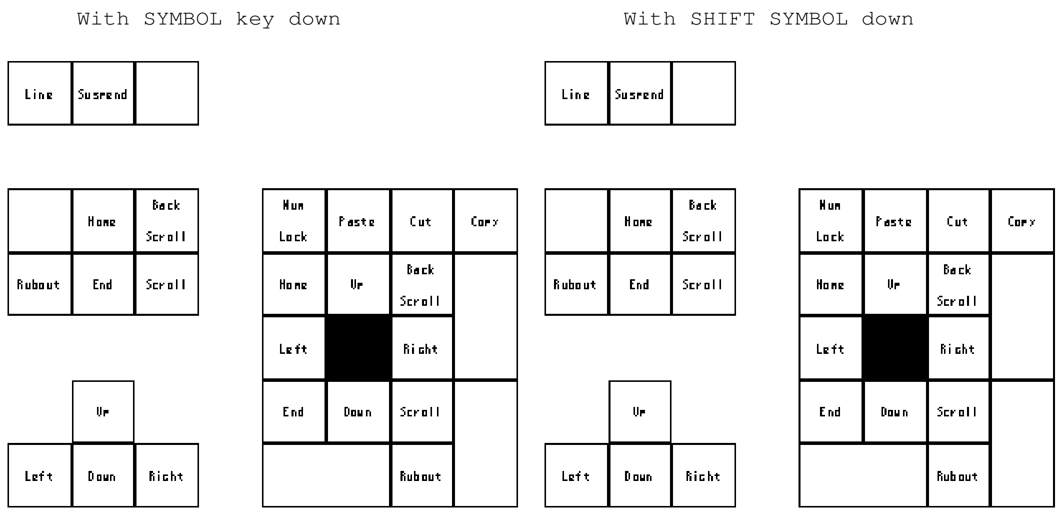

Figure 126. The with the Symbol Key as Modifier

Graphically displays the key mappings for the specified keyboard.

keyboard-layout The type of keyboard to display. Recognized keyboards are:

Apple	Mac 512K
Mac Portable ISO	Sun Type 4
Apple Extended	Mac 512K International
NCD N-101	Symbolics
Apple ISO	Mac Plus
SGI Iris	
Apple ISO Extended	Mac Portable
Sun Type 3	

keywords	:Include Codes, :Include Legends, :Include Mappings, :More Processing, :Output Destination
:Include Codes	{No, Octal, Decimal, Hex} Includes hardware mapping codes. The default is No.
:Include Legends	{Yes or No} Includes a page of the actual keytop legends. The default is Yes.
:Include Mappings	{Yes or No} Includes pictures of the variously shifted mappings. The default is Yes.
:More Processing	{Default, Yes, No} Controls whether **More** processing at end of page is enabled during output to interactive streams. The default is Default. If No, output from this command is not subject to **More** processing. If Default, output from this

command is subject to the prevailing setting of **More** processing for the window. If Yes, output from this command is subject to **More** processing unless it was disabled globally (see the section "FUNCTION M").

:Output Destination	{Buffer, File, Kill Ring, None, Printer, Stream, Window} Where to redirect the typeout done by this command. The default is the stream ***standard-output***.

Hardcopy Keyboard Layout from the Keyboard Control Activity (Select Activity

Keyboard Control) permits landscape printing. This is a simple way to make a keyboard template.

Show X Keyboard Mapping Command

Show X Keyboard Mapping *screen keywords*

Displays detailed information about the translation between X Window System keycodes and Genera keystrokes.

screen	Specifies the screen.
keywords	:All, :Match
:All	Shows all mappings, instead of just the nontrivial ones.
:Match	Shows mappings for the Genera keystrokes matching the specified substring.

The Keyboard Character Set

Some keyboards contain interesting keys that aren't in the standard Genera character set. For example, the Sun Type 4 keyboard contains keys labeled Undo, Cut, Copy, Paste, and arrow keys. These keys and others are translated into a special character set called the Keyboard character set, so that commands may be attached to them. Characters in the keyboard character set may be referred to using the `#\Keyboard` syntax (for example, `#\Keyboard:Cut`). This mechanism is used to support the arrow keys in Dynamic Windows and Zmacs, to make the Undo, Cut, Copy, and Paste keys perform their respective operations using the Cut Buffer feature of the X Window System, and to connect the PrintScreen key to the screen hardcopy feature.

The keyboard character set currently contains the following characters:

Cut	Paste	Copy	Undo
Again	Find	Print	
Left	Right	Up	
Back-Scroll	Home	Down	

Font Support in the Genera X Client

The Genera user interface can use either Genera fonts or the standard 75dpi X fonts. If the Genera fonts are available on the X server, it uses them. If they are not, but the standard 75dpi X fonts are available, it uses them. If neither the Genera fonts nor the standard 75dpi X fonts are available, Genera fonts are used in pixmap mode.

Using Genera Fonts in the Genera X Client

The Genera user interface can use the same character fonts whether it is displayed on the local console of a Symbolics workstation or on the console of another host using the X Window System. In the X Window System, these fonts may be displayed on the screen using two different mechanisms:

- The X server can be requested to draw a string using a native representation of the Symbolics font.
- The glyph image of each character in the Symbolics font can be sent over to the server as an X pixmap; the X server can be asked to copy that pixmap to the screen.

Both of these mechanisms provide the same results, but converting the fonts to the native representation used by the X server host is generally much faster.

The Genera X client supports both of these techniques. If the X server has access to a font of the same name as the corresponding Genera font, characters in that font are drawn using native operations. Otherwise, a pixmap is constructed. The Open Genera installation process installs native versions of Genera's fonts in the proper directory for access by the OSF X Server. The Symbolics software distribution includes BDF representations of all the Genera fonts, which may be compiled into the server's native format.

Using Standard 75dpi Fonts in the Genera X Client

The Genera user interface can use the standard 75dpi X fonts on X consoles. It chooses the following X font families from the character styles:

Character style family	*X font families*
FIX	Courier
SWISS	Helvetica
DUTCH	Times
JESS	Helvetica

The Genera X Client automatically maps between the Genera character set and the ISO8859 character set used by the stardard 75dpi X fonts, automatically choosing alternate glyphs from either the Symbol X font family or the corresponding Genera font drawn with pixmaps.

Color Support in the Genera X Client

If the X server host has color display hardware, you can draw in color by using the **:color** or **:gray-level** arguments to the **graphics:draw-***xxx* functions. The value of **:color** is a symbol that names a color (one of **:black**, **:red**, **:green**, **:blue**, **:cyan**, **:yellow**, **:magenta**, **:white**), a list (*red green blue*) where each element is a number between 0 and 1, inclusive, or a color object created by **color:make-color**. The value of **:gray-level** is a number between 0 and 1, inclusive. See the section "Pattern Options". See the option **:color**. See the option **:gray-level**.

See the function **color:make-color**.

Off-Screen Bitmaps in the Genera X Client

The Genera window system stores images in *bitmaps*: two-dimensional arrays of bits. Sometimes these are called *pixmaps* (two-dimensional arrays of pixels), especially when there is more than one bit per pixel, as on color screens. Each bitmap has a *width*, a *height*, and a *depth*; the depth is the number of bits per pixel.

A bitmap can be on the screen or off the screen. An on-screen bitmap is stored in special memory associated with the display hardware, so its contents are visible on the screen. The image of an exposed window resides in an on-screen bitmap so you can see it.

All other bitmaps are off-screen. The Genera window system often uses an off-screen bitmap to save the image of a deexposed window, so that the image can be quickly restored when the window is exposed, by copying the off-screen bitmap into an on-screen bitmap. This feature is under the control of the individual window, through the **:save-bits** option; however, most standard windows enable it. The Genera window system also uses an off-screen bitmap to save the image underneath a temporary window, such as a pop-up menu. User programs can also use off-screen bitmaps for their own purposes, using the facilities described below.

In platforms with remote window systems, such as MacIvory or the Symbolics UX, bitmaps have one more degree of freedom: a bitmap can be in Ivory's normal virtual memory, or it can be in host memory. On-screen bitmaps are always in host memory, because the host owns the display hardware. This is why the screen-array bitmap of an exposed window generally cannot be directly accessed by an Ivory program in an embedded system. Similar considerations apply to X Windows, substituting X Window Server memory for host memory.

Off-screen bitmaps can be in either Ivory memory or host memory. There are performance tradeoffs associated with the decision of where to store an off-screen bitmap. The principal constraints are the limited speed for copying images between Ivory memory and host memory, and the limited size of host memory.

On the one hand, placing an off-screen bitmap in host memory yields the maximum speed when copying images to or from the screen. Making the window server solely responsible for the copying is significantly faster than having to coordinate two processors and copy through the communications medium.

On the other hand, placing an off-screen bitmap in Ivory memory avoids consuming the limited amount of available server memory. The size of a bitmap depends on the application and on the size and depth of the display monitor. Some hosts, particularly X terminals or workstations with high-resolution color displays, may not have enough memory to store more than a few off-screen bitmaps.

These considerations mean the decision of whether to place an off-screen bitmap in Ivory memory or in host memory depends on the application and on the hardware configuration. The Genera window system decides as follows: Off-screen bitmaps for temporary use always go in host memory, unless not enough host memory is available even after swapping out other bitmaps. In this case they go in Ivory memory. Off-screen bitmaps for deexposed windows go in host memory or Ivory memory depending on how much host memory is available.

For further information:

See the macro **tv:with-off-screen-drawing**.
See the function **tv:%screen-allocate-sheet-temporary-bit-array**.
See the function **tv:%screen-deallocate-sheet-temporary-bit-array**.
See the macro **tv:with-temporary-sheet-bit-raster**.

Using Off-Screen Bitmaps in Your Application

Several interfaces are provided through which your application can use off-screen bitmaps. In embedded systems, all of these interfaces can store the off-screen bitmaps in host memory.

If you are presently doing things such as drawing to bit arrays and then **bitblt**ing to the screen, you should seriously consider using off-screen bitmaps in host memory, due to the substantial performance advantage in embedded systems such as MacIvory or the UX.

The available interfaces are:

tv:with-off-screen-drawing
tv:with-output-to-bitmap-stream
tv:with-output-to-bitmap
tv:allocate-bitmap-stream
tv:with-temporary-sheet-bit-raster

Using Off-Screen Bitmaps for Instantaneous Updates

This simple interface gives the appearance of an instantaneous update, rather than of each item appearing to be drawn separately. **tv:with-off-screen-drawing** works by copying the contents of a window to an off-screen bitmap, drawing into the bitmap, and copying back. In embedded systems, the off-screen bitmap always resides in host memory, provided sufficient host memory is available.

For instance,

```
(defun off-screen-strings ()
  (tv:with-off-screen-drawing (*terminal-io*)
    (dotimes (i 15)
      (write-string "12345678901234567890")
      (terpri))))

(defun off-screen-graphics ()
  (tv:with-off-screen-drawing (*terminal-io*)
    (graphics:with-room-for-graphics ()
      (graphics:draw-circle 100 100 50)
      (graphics:draw-triangle 20 0 120 0 70 75 :alu :flip))))
```

This involves some overhead in that you need to allocate and deallocate an off-screen bitmap, and copy from the window to the bitmap and back, each time the screen is visibly updated. In some cases the copy from the window to the bitmap is unnecessary, because the entire contents of the window will be redrawn. The **:complete-redisplay** option allows for optimization of this case. For example,

```
(defun off-screen-graphics-2 ()
  (tv:with-off-screen-drawing (*terminal-io*
                               :complete-redisplay t)
    (graphics:with-room-for-graphics ()
      (graphics:draw-circle 100 100 50)
      (graphics:draw-triangle 20 0 120 0 70 75 :alu :flip))
    (send *terminal-io* :refresh-margins)))
```

This is faster, but the previous content of the window is erased, where the previous example just scrolled it upwards.

Using Off-Screen Bitmap Streams

tv:with-output-to-bitmap-stream is a powerful interface that gives you an off-screen bitmap that supports all of the stream output and graphical operations of windows. The **:host-allowed** keyword controls whether the off-screen bitmap is stored in host memory or Ivory memory. There is no automatic copying from a window to the bitmap and back; you use the **:bitblt** and **:bitblt-to-sheet** messages to do this if and when you need it. Note that **:host-allowed t** is ineffective unless you also use **:for-stream** to specify a stream that leads to a screen; otherwise **tv:with-output-to-bitmap-stream** doesn't know which of potentially many hosts to use.

For example,

```
(defun off-screen-lines-1 (&aux (stream *terminal-io*))
  (fresh-line stream)
  (multiple-value-bind (cx cy)
      (send stream :visible-cursorpos-limits)
    (tv:with-output-to-bitmap-stream (bitmap-stream
                                      :for-stream stream
                                      :host-allowed t)
      (loop for x below 100 by 10 do
        (send bitmap-stream :clear-window)
        (loop for y downfrom (- 100 x) by 2 repeat 5 do
          (loop for x from x by 2 repeat 5 do
            (send bitmap-stream :draw-line x 0 0 y)))
        (send bitmap-stream :bitblt-to-sheet boole-xor 100 100 0 0
               stream (+ cx 150) (+ cy 150))))))
```

The previous example used a bitmap the full size of the window. We can improve it by creating a bitmap only as large as we need:

```
(defun off-screen-lines-2 (&aux (stream *terminal-io*))
  (fresh-line stream)
  (multiple-value-bind (cx cy)
      (send stream :visible-cursorpos-limits)
    (tv:with-output-to-bitmap-stream (bitmap-stream
                                      :for-stream stream
                                      :host-allowed t
                                      :width 100 :height 101)
      (loop for x below 100 by 10 do
        (send bitmap-stream :clear-window)
        (loop for y downfrom (- 100 x) by 2 repeat 5 do
          (loop for x from x by 2 repeat 5 do
            (send bitmap-stream :draw-line x 0 0 y)))
        (send bitmap-stream :bitblt-to-sheet boole-xor 100 100 0 0
               stream (+ cx 150) (+ cy 150))))))
```

If you specify values for **:width** and **:height** that are too small, the bitmap automatically expands so it is at least large enough to hold all the bits drawn. Of course the expansion takes time, so it is preferable to specify accurate values for **:width** and **:height**.

We can further improve the speed by using **:draw-multiple-lines** in place of **:draw-line**, to cut down on overhead. It speeds up this particular example by only 7 percent since the line-drawing speed of the Macintosh is the limiting factor in either case.

```
(defun off-screen-lines-2-m (&aux (stream *terminal-io*))
  (fresh-line stream)
  (multiple-value-bind (cx cy)
      (send stream :visible-cursorpos-limits)
    (tv:with-output-to-bitmap-stream (bitmap-stream
                                      :for-stream stream
                                      :host-allowed t
                                      :width 100 :height 101)

     (loop for x below 100 by 10 do
        (send bitmap-stream :clear-window)
        (send bitmap-stream :draw-multiple-lines
          (loop for y downfrom (- 100 x) by 2 repeat 5
                nconc
                  (loop for x from x by 2 repeat 5
                        collect x collect 0
                        collect 0 collect y)))
        (send bitmap-stream :bitblt-to-sheet boole-xor 100 100 0 0
              stream (+ cx 150) (+ cy 150))))))
```

tv:with-output-to-bitmap is similar to **tv:with-output-to-bitmap-stream** with the additional feature that it copies the off-screen bitmap into a Lisp array and returns it. Use this when you wish to capture the result of some output operations in an array of bits, rather than (or in addition to) displaying the result on the screen.

If you need an off-screen bitmap stream with a more permanent lifetime, you can use explicit allocation and deallocation. See the function **tv:allocate-bitmap-stream**. See the function **tv:deallocate-bitmap-stream**. For example,

```
(defun off-screen-lines-3 (&aux (stream *terminal-io*))
  (fresh-line stream)
  (multiple-value-bind (cx cy)
      (send stream :visible-cursorpos-limits)
    (let ((bitmap-stream (tv:allocate-bitmap-stream
                                      :for-stream stream
                                      :host-allowed t
                                      :width 100 :height 101)))
      (loop for x below 100 by 10 do
        (send bitmap-stream :clear-window)
        (loop for y downfrom (- 100 x) by 2 repeat 5 do
          (loop for x from x by 2 repeat 5 do
            (send bitmap-stream :draw-line x 0 0 y)))
        (send bitmap-stream :bitblt-to-sheet boole-xor 100 100 0 0
              stream (+ cx 150) (+ cy 150)))
      (tv:deallocate-bitmap-stream bitmap-stream))))
```

In a more realistic example, the calls to **tv:allocate-bitmap-stream** and **tv:deallocate-bitmap-stream** would be in two different functions; otherwise **tv:with-output-to-bitmap-stream** would work just as well.

The bitmap stream can become invalid if the window system is shut down and started again. It is best not to retain an off-screen bitmap stream permanently, but only for the duration of one operation. To minimize the overhead of allocating and deallocating a bitmap stream, you can allocate it when your application window is exposed and deallocate it when your application window is deexposed.

Using Off-Screen Bitmaps with Low-level Drawing Primitives

If you only need a bitmap, without the stream output and graphical operations of windows, either because you are using the low-level drawing primitives, or because you only do **bitblt**'s, you can use the lower-level allocation primitives. Note that if you are using **sys:%draw-***xxx*, you need to switch to **tv:sheet-draw-***xxx*, since the system must be given a sheet in order to find the screen that connects to the host. There is no significant overhead in this switch.

```
(defun off-screen-lines-4 (&aux (sheet *terminal-io*))
  (tv:with-temporary-sheet-bit-raster (bitmap sheet 100 100)
    (loop for x below 100 by 10 do
      (tv:sheet-force-access (sheet)          ;Make sure has a screen array
        (letf (((tv:sheet-screen-array sheet) bitmap))  ;Use this one temporarily
          (tv:sheet-draw-rectangle 100 100 0 0 boole-andc1 sheet)
          (loop for y downfrom (- 100 x) by 2 repeat 5 do
            (loop for x from x by 2 repeat 5 do
              (tv:sheet-draw-line x 0 0 y boole-ior nil sheet))))
        (tv:sheet-bitblt boole-xor 100 100 bitmap 0 0 nil 200 200 sheet)))))
```

If you need an off-screen bitmap with a more permanent lifetime, you can use explicit allocation and deallocation. Instead of using **tv:with-temporary-sheet-bit-raster**, you can call **tv:%screen-allocate-sheet-temporary-bit-array** and **tv:%screen-deallocate-sheet-temporary-bit-array**. Note that all these programs work on the XL and the 3600 series as well.

tv:allocate-bitmap-stream &key *:for-stream :host-allowed :width :height :bits-per-pixel :graphics-transform* *Function*

Allocates an off-screen bitmap stream. You can perform textual and graphic output operations on the stream returned. The results will not be visible, since the bitmap is off-screen. You can use the **:bitblt-to-sheet** message to make the result visible by copying from the off-screen bitmap to a window.

The bitmap stream can become invalid if the window system is shut down and started again. It is best not to retain an off-screen bitmap stream permanently, but only for the duration of one operation. To minimize the overhead of allocating and deallocating a bitmap stream, you can allocate it when your application window is exposed and deallocate it when your application window is deexposed.

:for-stream A stream that outputs to a related window. This provides defaults for the width, height, depth, and coordinate transformation; if **:host-allowed t** is specified, the host owning the window's screen will be used.

:host-allowed	True if the off-screen bitmap should be stored in host memory if possible. The default is false, which always uses Ivory memory. **:host-allowed t** only works if **:for-stream** is specified.
:width	The initial width of the bitmap. It expands if necessary.
:height	The initial height of the bitmap. It expands if necessary.
:bits-per-pixel	The depth of the bitmap.
:graphics-transform	A coordinate transformation. The default is the stream's transform if **:for-stream** is specified. Otherwise, the identity transform is the default.

See the message **:bitblt-to-sheet**.

See the function **tv:with-output-to-bitmap-stream**.

tv:deallocate-bitmap-stream *bitmap* *Function*

Deallocates an off-screen bitmap stream when you are no longer using it. See the function **tv:allocate-bitmap-stream**.

:bitblt-to-sheet *alu width height x y sheet sheet-x sheet-y* *Message*

Sends this message to an off-screen bitmap to copy its contents onto a window where it will be visible. This performs a **bitblt** from the *width* by *height* rectangle of the bitmap whose top-left corner is at *(x,y)* to the *width* by *height* rectangle of the window *sheet* whose top-left corner is at *(sheet-x,sheet-y)*.

See the function **bitblt**.

See the function **tv:allocate-bitmap-stream**.

See the function **tv:with-output-to-bitmap-stream**.

tv:with-output-to-bitmap *(*&optional *stream* &key *:for-stream :graphics-transform)* &body *body* *Function*

stream	The stream to which to return the bitmap.
:for-stream	The stream for which the bitmap is intended.
:graphics-transform	An optional transform to be applied.

Returns a raster array and positions containing the image output by *body*.

```
(defun bitmap-example (&optional (stream *standard-output*))
  (graphics:with-room-for-graphics ()
    (graphics:draw-triangle 0 0 200 0 50 50
         :tile (tv:with-output-to-bitmap (bstream
                     :for-stream stream)
                (graphics:draw-circle 0 0 10
                     :gray-level .25 :stream bstream)
                (graphics:draw-regular-polygon 8 0 16 0 6
                       :gray-level .75
                     :stream bstream)))))
```

tv:with-output-to-bitmap-stream (*bitmap-stream* &rest *args* &key (*for-stream* **nil**) &allow-other-keys) &body *body* *Function*

bitmap-stream A stream that is a raster array intended to hold the image generated by *body*.

args **:for-stream**, the stream for which the bitmap is intended, and, optionally, **:graphics-transform**, an optional transform to be applied.

Binds *bitmap-stream* to a specially allocated stream that accepts the graphic output during execution of *body*. At any time, the **:bitmap-and-edges** message to this stream returns the current image.

The Genera X Server

Introduction to the X Server

The Symbolics X Server program handles X protocol output and input requests from other systems on the network and performs the requested operations on the local system's screen. It allows you to operate X client applications (such as "xterm") running on other systems from your Symbolics Computer's console.

The Symbolics X Server is a port of version 4 of the standard MIT-supplied server, which is written in C. Little work has been done to improve the performance of the portable server.

Using the X Server

The X Server system defines an application called X11 Server, available by default on Select Square. (See the section "Customizing the SELECT Key" for a discussion of how to change this binding.) You can use the following commands to control the X11 Server application:

Start Server X Server Command

Creates a background process to manage X network connections.

Halt Server X Server Command

Kills the background process that manages X network connections.

Switch Mode X Server Command

Passes control of the keyboard and mouse to the X Server. When the X Server has control of the keyboard and mouse, all keystrokes and mouse motion will be passed through to the appropriate X client program. Therefore, the normal Genera utilities are unavailable until you exit this mode.

For example, if the current X application is the Remote Screen program, Select L causes a Lisp Listener to be selected within the remote screen, rather than selecting your "local" Lisp Listener.

To return control to Genera (and get back to the command loop of the X11 Server) you can press the Network key.

Halt X Server Command

Kills the background process that manages X network connections.

Start X Server Command

Creates a background process to manage X network connections.

Once you've started the X Server it is possible for remote applications to create windows on your Symbolics Computer's screen (within the X11 Server application's display area).

Compiling Fonts for the X Server

Normally you won't need to compile any fonts, as the standard set of X fonts are included in the X Server distribution, but if you have local fonts in Bitmap Distribution Format, you can compile them to the Server Native Format used by the X Server.

Compiling BDF Files to Symbolics X Server SNF Files

The BDFtoSNF program is used to convert X fonts from the standard Bitmap Distribution Format (BDF) to the Server Native Format (SNF) used by the X Server.

The BDFtoSNF system can be invoked through the "Compile File" command:

```
Command: Compile File (file) sys:x11;fonts;local;localfont.bdf
```

Create X Font Directory Command

The MkFontDir program is used to build an X Server fonts.dir file for a directory of SNF fonts. A new fonts.dir file has to be created every time a SNF font file is created or deleted.

Use the "Create X Font Directory" command to invoke the MkFontDir program:

```
Command: Create X Font Directory (for directory) sys:x11;fonts;local;
```

You must use a logical pathname as the directory name for the Create X Font Directory command.