Open Genera User's Guide

Introduction to Open Genera

Open Genera makes Symbolics' Genera software environment available to DEC Alpha workstation users. The host workstation runs its standard operating system software Digital Unix (originally called OSF/1 [which was the Open Software Foundation's POSIX-compliant UNIX implementation]) augmented with software to provide Genera. Open Genera negotiates with the host operating system to provide Genera's I/O services using the host's peripherals, and to present Genera's user interface using the host's console.

As an integrated system, Open Genera offers the user more than just a choice between the UNIX and Genera environments; it integrates Genera with the host operating system, particularly in the areas of user interface and networking. Genera applications appear within the host window system just as native Digital Unix applications do, and a wide variety of network services are provided for communication between the two systems. Open Genera adds a new dimension to the UNIX environment, and expands Genera's potential by providing ready access, for both users and applications, to existing UNIX software facilities.

Open Genera is available with the complete Genera system for software development. The development system provides compatibility with Genera applications developed on Symbolics 3600-family, XL-family, NXP1000 and MacIvory-family workstations.

Beginning with release 2.0 of Open Genera, multiple copies of Genera can be running simultaneously on the same DEC Alpha workstation. Each copy simulates an independent workstation environment. This permits developing and testing multi-host distributed Genera applications on a single piece of hardware.

Notes About This Document

This manual is intended for use in conjunction with the Genera documentation set. Please consult the online Genera documentation for more detailed information on Genera.

Throughout this document, the term "UNIX" is interchangeable with the term "Digital Unix" (which is a flavor of UNIX). However, a reference to "Digital Unix" does not necessarily apply to other variants of UNIX.

The older term for Digital Unix, namely "OSF/1" might still appear in some of the documentation, if so it has the same meaning as "Digital Unix".

Open Genera Architecture

Open Genera relies on the host workstation and its peripherals to provide the basic I/O services needed to support Genera. These services include bootstrapping support, disk access for storing Genera world and paging files, and a connection to the host network interface. This layer of software is called Genera Life Support.

Open Genera consists of three main parts: the interpreter, life support, and the Genera Software Engineering Environment (SEE).

Open Genera shares the network interface of its host, and appears on the host's network just like any other system. Although the host workstation and the "virtual coprocessor" share a single Ethernet hardware address, they use separate *protocol*, or software addresses, and therefore appear as separate hosts on the network. Other hosts request network services from either Digital Unix or Genera independently, using the appropriate protocol addresses.

If you run more than one copy of Open Genera on the same Alpha workstation, each copy has its own protocol and network addresses. This permits developing and testing distributed applications on a single machine.
The Open Genera SEE offers extensive networking facilities and adapts easily to heterogeneous, multivendor environments. It includes protocols and services to support many different operating systems, and automatically chooses the best protocols, hosts, and routes to obtain a requested service. Open Genera software distribution includes a number of protocols specifically to enhance communication with UNIX, including:

IP/TCP	The ARPA Internet protocol family
NFS	A standard UNIX file transfer protocol
RPC	A compatible version of Sun's Remote Procedure Call protocol
X11	The X Window System display protocol

Like all Digital Unix application programs, Open Genera uses Digital Unix swapping partitions for Genera paging space. The Genera world-load file is stored as a Unix file and can reside on any NFS-accessible file server. If the world-load file is not on the local machine, performance during boot and access of world pages will depend on network load.

Open Genera User Interface Architecture

Open Genera is an X client that communicates with an X server running on the host workstation for its user interface. In fact, Open Genera can be used from any host on its network that supports an X server. The default is the local host machine. Open Genera may be used without a user interface, to provide its usual network services or to function as a "back end" to a UNIX application using remote procedure call (RPC) technology. In normal operation, however, Genera presents its user interface on the host's console, using the X Window System developed at the Massachusetts Institute of Technology.

The Genera user interface may be presented in the form of the standard Genera console framework, which includes a tiled window manager with an extensive status display, and provides mechanisms for selecting between the available Genera activities.

In all cases, the Open Genera user interface is programmed and managed by the same Genera user interface facilities used in other Symbolics workstations. The X

Window System is used as an implementation mechanism only; its programming interfaces and toolkits are not supported on Open Genera, although the host operating system may provide them for use with the same X server.

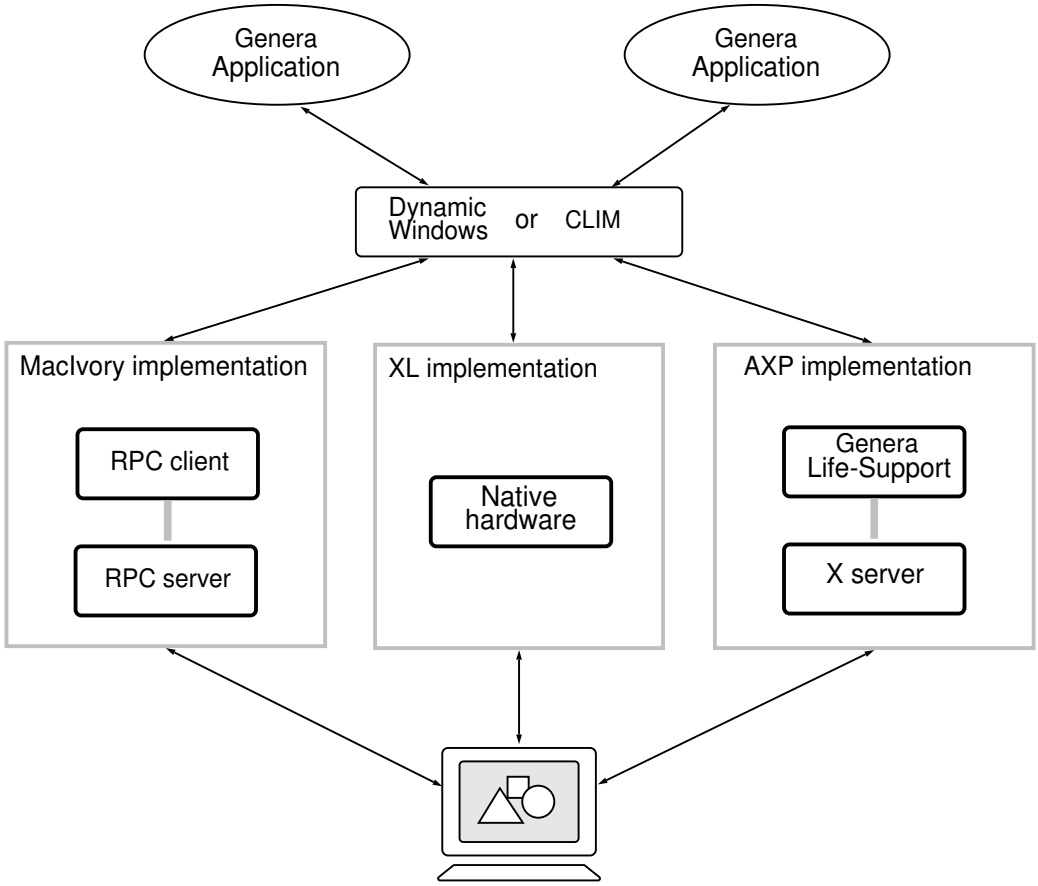

Figure 1. Open Genera User Interface Architecture

Figure 1 illustrates the architecture of the Genera user interface implementation. The two programmer interfaces CLIM and Dynamic Windows interface are supported. These facilities are implemented using three different technologies:

- In Symbolics 3600-series and XL workstations, the implementation directly manipulates the Symbolics console.

- In Symbolics MacIvory systems, the implementation uses an RPC protocol between the coprocessor and the host Macintosh, which manipulates the console hardware using its QuickDraw software library.

- In Open Genera and UX-family systems, the implementation uses the X protocol; an X server on the host system manipulates the console hardware and shares access among multiple X clients.

Open Genera uses facilities of the X protocol to define a mapping between the keys available on the host keyboard and the keys used by the Genera SEE. On a DEC Alpha workstation , only the DEC LK401 AA model keyboard has the necessary keys to implement this mapping. See the section "Open Genera Keyboard Support".

Getting Started with Open Genera

Before you can use Open Genera, it must be properly installed and configured. Installation consists of the following steps:

- Installing Open Genera software
- Configuring the Digital Unix kernel and swap space
- Installing the Genera software and world loads
- Configuring the Digital Unix and Genera networks for the new machines
- Installing the Genera source code and documentation

See the section "Installing Open Genera" for a detailed description of the Open Genera installation process.

When you are ready to proceed, boot the DEC Alpha workstation. Procedures for this will vary, but if the workstation is turned off, turning it on will usually suffice to boot it. Note that Digital Unix caches modified copies of critical data structures in volatile memory. You should not shut down the system by turning off the power. Talk to your site administrator if you have any questions about operating your workstation.

At this point, the Digital Unix system should be waiting for you to login. If it was configured by DEC, the system will start a window system immediately after booting. A window will be displayed, prompting for a user name and a password.

Log in to the DEC Alpha workstation.

Run the `genera` Digital Unix application after logging in to get Genera to boot. Genera will boot and bring up both an iconified Cold Load stream and Genera's user interface, a large X window with an initial display showing the contents of the world and some other information. (The Cold Load stream window is not required during normal Genera operation, but don't destroy it. It is normally iconified, and will de-iconify automatically if necessary. You may reposition or bury it, but don't destroy it as it is used a as last resort for communicating with Lisp in the face of serious X-server, network, or internal errors.)

You may now log in to Genera and use the system much as you would a Symbolics workstation, albeit with a different keyboard. If you are unfamiliar with Genera, see the document *Genera Workbook* for an introduction. Note that the Genera console image is just one X window among many in the DEC Alpha workstation's user

interface; it may be moved, buried, or iconified at will as you switch between Genera and Digital Unix applications.

The proper way to close the Genera window is to abort the `genera` Digital Unix program following these steps:

1. Specify `Halt Genera` from a Lisp listener.

2. Answer `YES` to the prompt that appears.

See the section "Halt Genera Command".

You may now log out of Digital Unix to complete your session.

Controlling Open Genera

In the Digital Unix environment, a program is loaded from the file system on demand into a separate address space, which is discarded when the program exits. A Genera environment consists of a single large virtual memory space, within which all the code and data objects of all loaded Genera applications exist.

Objects in Genera's virtual memory are stored as follows:

The original Genera world load (disk)
: A Genera world load is a complete image of a Genera environment, frozen in time at the moment the world was saved. It may be composed of several files using the incremental disk save feature (IDS). A world load is never modified after it is saved.

As Genera runs, it acquires additional memory resources for new and modified code and data structures by negotiating with Digital Unix for swap space, just as any other application program does. Information stored in swap space is not preserved across sessions.

The main memory of the DEC Alpha
: All objects in a Genera environment are stored on disk in either the world or Digital Unix swap space. However, recently used objects are copied from the slow disk to the fast main memory, to improve performance.

The Digital Unix `genera` program is used to boot Genera and to invoke Genera applications.

The `genera` Digital Unix Application

The Digital Unix command interface for operating Open Genera is an application named `genera`. Parameters controlling the operation of genera, including which world load it boots, are given in any of several configuration files and/or in command options. The command options are described in the following section. A description of the equivalent syntax for the `.VLM` configuration files is also provided.

genera *options*

options

-spy no -spy yes
: Specifies whether or not the Spy is enabled. Default is -spy no. If you use -spy yes and you specified an IP address for your Open Genera, this address will be used as the diagnostic host's address. Otherwise, you must specify a diagnostic host.

-diagnostic HOSTNAME
: Specifies the host which will be used for remote debugging. This argument must be specified if the Spy is enabled. An appropriate choice is the name of your Alpha.

-world PATH
: Specifies the pathname of the Lisp world to be loaded into Open Genera's memory and booted. The default is

```
/usr/lib/symbolics/Genera-8-5.vlod
```

-debugger PATH
: Specifies the pathname of the VLM Debugger to be loaded into the VLM's memory and invoked should Lisp crash. The default is

```
/usr/lib/symbolics/VLM_debugger
```

-network SPECIFICATION -network UNIT,SPECIFICATION
: Specifies a network interface, that is, the way Open Genera will be accessed from the network. The first form defines the main network interface (unit 0). The second form is used to define additional interfaces; `UNIT` must be a number between 0 and 7. (You can specify the main interface using either syntax.) At least one interface must be defined. `SPECIFICATION` can be either the name of the Open Genera host or any string acceptable to the IFEP's `:Set Network Address` command (for example, `ETA-FISH`, `INTERNET|128.81.41.48`, `CHAOS|24460`, etc.).

-vm NNN
: Specifies how much of your Alpha's swap space should be used by Lisp where `NNN` is the number of megabytes to be used by Lisp. This specification includes the space occupied by the world. The default memory limit is 200 megabytes (MB) or 40 megawords (MW). If Lisp exceeds your specification, it will drop into the VLM Debugger with an appropriate message. If you *:Continue*, genera increases your limit by 25%.

-ids yes -ids no
: Specifies whether the ability to save IDS (incremental disk save) worlds is enabled or not. The default is that this ability is disabled (no). (Note that it is always possible to *boot* an IDS world regardless of this option.)

-searchpath PATH
: When asked to boot an IDS, the genera command must search for the IDS' ancestor worlds. The genera program searches the

directory containing the IDS and then the directories specified in the world file search path. The general form of PATH is a colon-separated list of directory names, for example:

```
/usr/a/smbx:/usr/b/smbx
```

The default world file search path is

```
/var/lib/symbolics:/user/lib/symbolics
```

Also, you can override the default search path by setting the WORLDPATH shell environment variable. The syntax of the search path is a colon-separated list, as shown above. The directories in the path are searched from left to right.

The following arguments control the main X console for Genera.

Options which have abbrevations are mentioned in parentheses after the full option name.

`-display DISPLAY` The default for `-display` is the display specified by the `DISPLAY` environment variable. If that variable isn't set, the local host is used.

`-geometry GEOMETRY`

`-iconic yes -iconic no`
The default for `-iconic` is `no` (that is, the window starts fully visible.)

`-foreground (-fg) COLOR`

`-background (-bg) COLOR`

`-bordercolor (-bd) COLOR`

`-borderwidth (-bw) WIDTH`

The following standard X arguments control the cold load window. Again, abbreviations are in parentheses.

`-coldloaddisplay (-cld) DISPLAY`
The default for `-coldloaddisplay` is the same as for `-display`.

`-coldloadgeometry (-clg) GEOMETRY`

`-coldloadiconic (-cli) yes -coldloadiconic (-cli) no`
The default for `-coldloadiconic` is `yes` (that is, the window starts as an icon).

`-coldloadforeground (-clfg) COLOR`

`-coldloadbackground (-clbg) COLOR`

`-coldloadbordercolor (-clbd) COLOR`

`-coldloadborderwidth (clbw) WIDTH`

In addition to command line arguments, you can specify the options to control genera in a configuration file. The syntax of a configuration file is the standard X resource file format. Each line of the file contains an option name and value separated by colons. The "!" character may be used to start a comment, which extends to the end of the line.

The `genera` command uses a succession of sources to establish and modify its option settings, arriving at the set finally used in booting and running Open Genera. The procedure is as follows, that is `genera`:

1. establishes its built-in default settings.

2. reads the configuration file `/var/lib/symbolics/.VLM`, if it exists, and uses any setings in it to supersede the corresponding values already established.

3. reads the `.VLM` file, if any, in your home directory and uses any settings from it to supersede corresponding values.

4. reads the `.VLM` file, if any, in the working directory and uses any settings from it to supersede corresponding values.

5. processes any options in the command line and uses them to supersede corresponding values.

A Sample .VLM File

A sample `.VLM` file is illustrated in the following example:

```
!
! Sample Configuration File
!
*network:               HOST-NAME

genera.enableIDS:       yes
genera.virtualMemory:   350
genera.world:           /var/lib/symbolics/scrc-8-4.vlod
```

An option that is prefixed with "genera." applies only to the genera command. An option that is prefixed with "*" applies to the genera command and any other commands which implement the VLM. The values for an option that appear in the file are the same as those that appear on the command line.

The option names used in the file and the corresponding command options are

Name in File	Name on Command Line
spy	-spy
diagnosticHost	-diagnostic
world	-world
debugger	-debugger

network	-network SPECIFICATION
virtualMemory	-vm
enableIDS	-ids
worldSearchPath	-searchpath
main.display	-display
main.geometry	-geometry
main.iconic	-iconic
main.foregroundColor	-foreground
main.backgroundColor	-background
main.borderColor	-bordercolor
main.borderWidth	-borderwidth
coldLoad.display	-coldloaddisplay
coldLoad.geometry	-coldloadgeometry
coldLoad.iconic	-coldloadiconic
coldLoad.foregroundColor	-coldloadforeground
coldLoad.backgroundColor	-coldloadbackground
coldLoad.borderColor	-coldloadbordercolor
coldLoad.borderWidth	-coldloadborderwidth

Exiting the genera Application

Use the Halt Genera command to pause or exit the genera application. Note that exiting the genera application is equivalent to powering off a "real" Lisp Machine: all your state is irretrievably destroyed. You must save any important state before exiting.

Halt Genera Command

Halt Genera *keywords*

keywords	:Logout :Shutdown :Query :Delay :Reason :Simulate
:Logout	{Yes or No} Default is Yes. Logging out before halting is recommended. If you have not already logged out, :Logout Yes will perform a Logout before halting. Specify :Logout No if you do not want to log out first.
:Shutdown	{Yes or No} Whether to exit the genera application after halting. Default is Yes. Usually, if you are through with Genera, you will want to exit the genera application to release the resources it is using and make them available to other Digital Unix applications. Note that exiting the genera application is equivalent to powering off a "real" Lisp Machine: all your state is irretrievably destroyed. You must save any important state before exiting.

:Query — {Yes, No or Confirm-Only} Whether to ask about each decision, just confirm, or don't ask at all. Default is Yes; mentioned default is Confirm-Only.

:Delay — {None or a Time Interval} How long to delay before halting. Default is 5 minutes if there are active servers, otherwise None. If Genera is serving other client processes, you can specify a delay before halting to allow those clients enough time to complete their processes.

:Reason — {A string} Default for "halting." Used to warn active clients about pending halt.

:Simulate — {Yes or No} Whether to just show the effectof this command without actually doing it. Default is No.

If you plan to come back to Open Genera shortly, or you do not want to lose your state, specifying :Shutdown No will pause the `genera` application, but retain Genera's state and you will be able to resume your Genera session by typing :Start to the VLM Debugger prompt in the VLM Debugger window. While the `genera` application is paused, it will not consume any CPU resources, although it will retain its swapping resources (where Genera's state it stored). To reclaim those resources (and exit the genera application altogether) you can type :Shutdown to the VLM Debugger prompt.

Using the Open Genera Debugger

Genera Life Support performs the initialization steps formerly associated with the Front-End Processor (FEP) of Symbolics machines. In order to cold boot Genera, restart the Digital Unix `genera` program (essentially throwing away it's memory image and restarting from the world load).

The Virtual Lisp Machine (VLM) Debugger is a debugger of last resort, used when Genera (Lisp) halts or is in an inconsistent state and unable to proceed. The VLM Debugger shares the Cold Load console with Genera.

Genera uses the Cold Load console as an alternative console when the main X Console is unusable for some reason. Usually, when Genera uses the Cold Load console, you are in the Genera debugger. A number of proceed options to work around the current problem are offered.

If Genera is unable to handle an error itself (because of internal inconsistencies, recursive errors, or insufficient resources) it will halt and the VLM Debugger will use the Cold Load console to display an error message.

Sometimes, you will be able to recover from the error. For example, if Genera would exceed the virtual image size you requested, it will halt and display a message advising you to use the VLM Debugger `:Continue` command if you want to proceed. The VLM Debugger is used because, unlike the Genera debugger, it does not "cons" (use additional memory resources) when such resources are already exhausted. These are called *proceedable* errors.

If an unexpected error occurs, or an error occurs that does not indicate any possibility of proceeding via `:Continue`, you can use the VLM Debugger debugging commands to examine the state of Lisp, perhaps determining the cause of the bug or at least gather enough information to submit a bug report.

Sometimes you will be able to recover from an unexpected error, or an error which does not have an option to continue, by using the VLM Debugger `:Start` command, which will cause Genera to "warm boot" (essentially, reinitialize itself, but in a way that preserves most, if not all, user state, such as modified files, buffers, mail, etc.) It is often worth retrying the `:Start` approach at least twice. The VLM Debugger `:Start` command will first try to recover all Genera processes, including the process with the error that caused Lisp to halt. If that is unsuccessful (you immediately halt again), a second :Start command will forcibly abort and reset the erring process (but no other processes).

On the VLM, there is another option for debugging system errors. If you enable the VLM remote debugger in your `.VLM` configuration file, and your Digital Unix host is accessible to the Internet, Symbolics Customer Support can attempt to remotely diagnose the system error. You must supply the network address of both the Digital Unix host and VLM for this service and you must have enabled the -spy option when you started the genera application. See the section "The Genera Digital Unix Application".

The VLM Debugger is nearly identical in function to the IFEP Debugger. It can be used by experienced users to diagnose bugs that halt normal operation of Lisp and its debugger. The VLM Debugger can also be used as a last resort to gather information for a bug report. If you are a former Ivory Lisp Machine user, you will find that the VLM Debugger is much the same as the IFEP Debugger, but is not a separate subcommand (there is no IFEP). Some few IFEP commands (Continue, Show Version, Shutdown, Start, Show Status, and Clear Screen) are available, in addition to all the IFEP Debugger commands and accelerators. For more information,see the section "Debugging on Ivory-based Machines". Note that the prompt for the VLM Debugger is `VLM Debugger:` (instead of the arrow prompt used by the IFEP Debugger.

Open Genera User Interface

Genera presents its user interface on the host's console, using the X Window System developed at the Massachusetts Institute of Technology. Open Genera is an X client that communicates with an X server running on the host workstation for its user interface. In fact, Open Genera may be used from any host on its network that supports an X server.See the section "Connecting to Genera".

The Genera user interface may be presented in the form of the standard Genera console framework, which includes a tiled window manager with an extensive status display, and provides mechanisms for selecting among the available Genera activities.

In all cases, the Open Genera user interface is programmed and managed by the same Genera user interface facilities used in other Symbolics workstations. The X

Window System is used as an implementation mechanism only; its programming interfaces and toolkits are not supported on Open Genera, although the host operating system may provide them for use with the same X server.

The X Window System provides a comprehensive set of graphic operations with well-defined scan conversion semantics that are compatible with those of the Genera graphics substrate. See the section "Scan Conversion". Most of Genera's graphic operations, from simple character and line drawing to more complicated shapes and rendering options, are simply translated into X requests and forwarded to the X server. Consequently, the performance of graphic operations is very dependent on the performance of the X server. Some X servers have been optimized for only the most common operations, and as a client Genera inherits that characteristic.

Significant performance problems in the Genera X Client are usually caused by excess traffic between the client and server, particularly sending bitmaps back and forth. Genera provides a number of mechanisms for minimizing such traffic, and uses them itself to improve performance.

Open Genera may be used without a user interface to provide its usual network services or to function as a "back end" to a Digital Unix application using remote procedure call (RPC) technology.

Connecting to Genera

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 Open Genera, Genera 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 X client supports multiple user interfaces simultaneously, possibly displayed on different consoles. This does not, however, make Genera a multiuser operating system. The Open Genera Software Engineering Environment 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.

Open Genera Keyboard Support

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.

Open Genera accommodates the keyboard on your DEC Alpha by translating keystrokes from the physical keyboard into its own abstract set of keystrokes. Only the DEC LK401 AA model keyboard has the necessary keys to implement this mapping. (Using other DEC keyboards, including the DEC LK421 AA keyboard, will make use of Open Genera difficult, if not impossible.) The DEC LK401 AA keyboard is shown in Figure!. The general-purpose function keys are mapped to Open Genera as shown in Figure !, Figure !, Figure !, and Figure !.

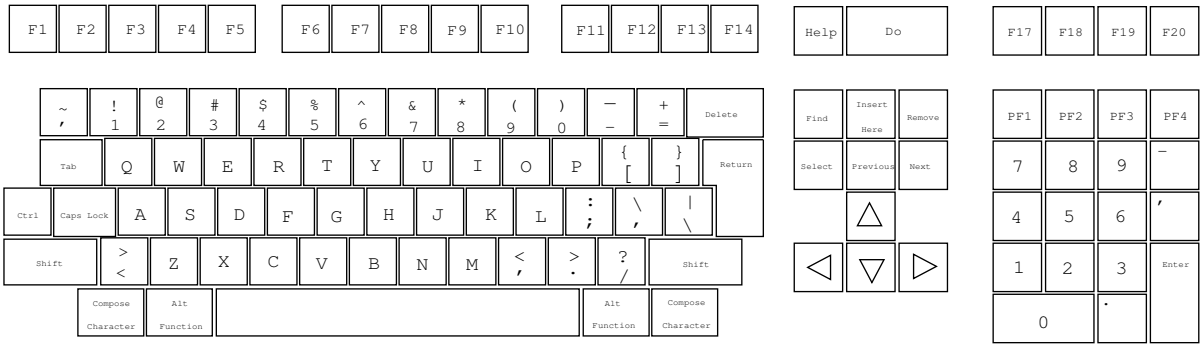

Figure 2. DEC LK401 AA Keyboard

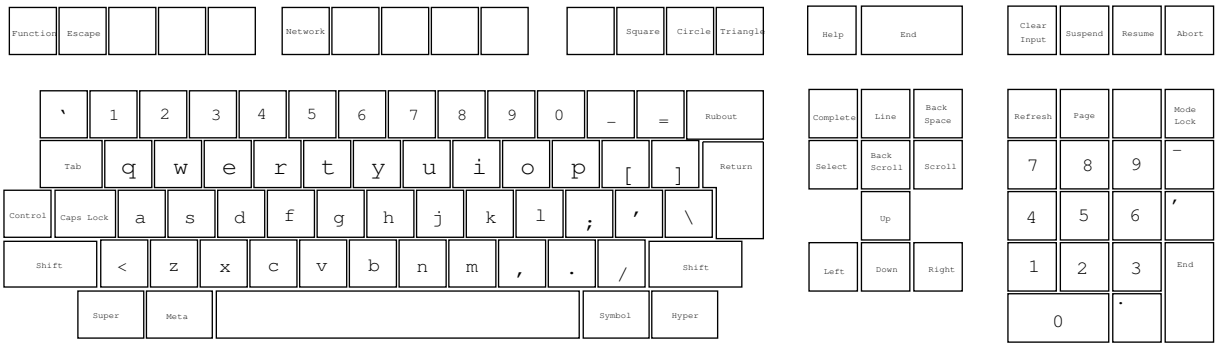

Figure 3. Open Genera Interpretation of the DEC LK401 AA Keyboard

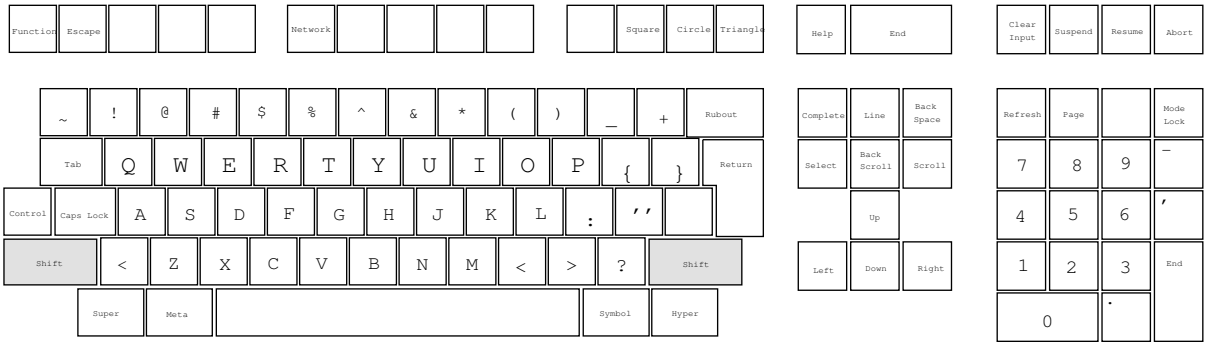

Figure 4. Open Genera Interpretation of the DEC LK401 AA Keyboard Using the Shift Key

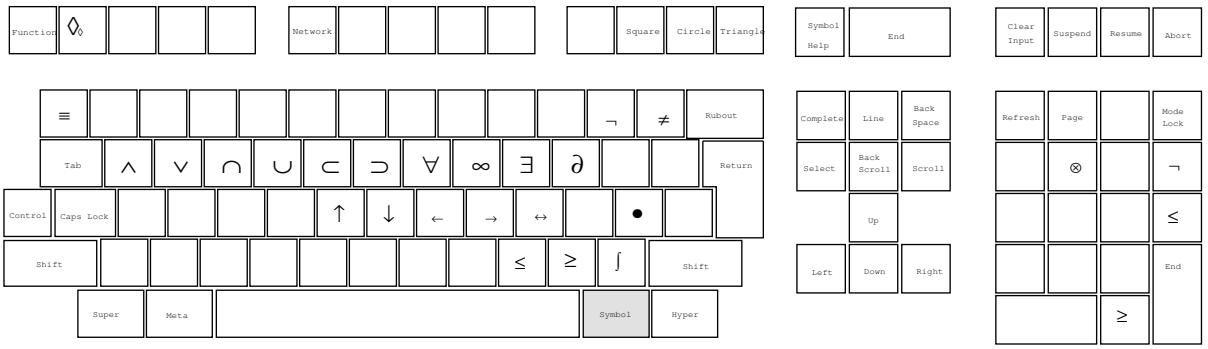

Figure 5. Open Genera Interpretation of the DEC LK401 AA Keyboard Using the Symbol Key

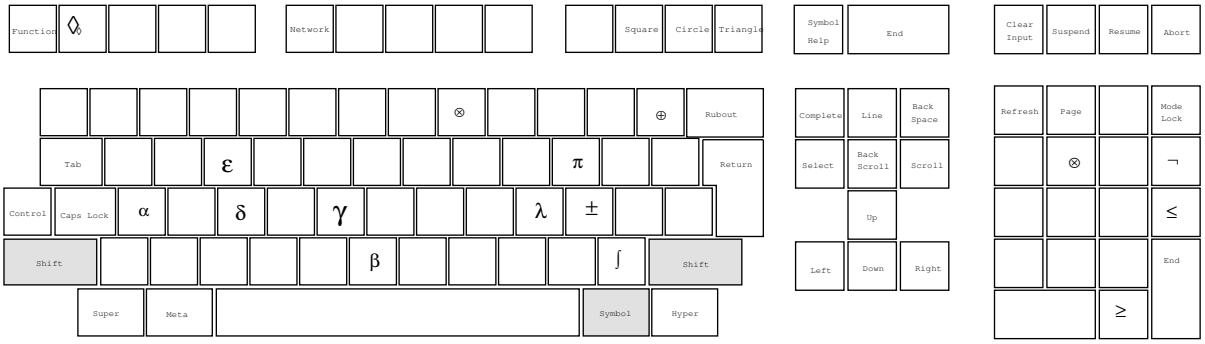

Figure 6. Open Genera Interpretation of the DEC LK401 AA Keyboard Using the Symbol and Shift Keys

Use the Show Keyboard Layout command for a graphic display of the key mappings (the default for the *keyboard-layout* argument is DEC LK401 AA). See the section "Show Keyboard Layout Command".

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**.

Alternatively, you can access the workstation (or X servers) color capabilities by using CLIM. See the section "Drawing in Color in CLIM".

File System Access on Open Genera

Genera provides access to many file systems using Symbolics' generic network system. For most DEC Alpha users, the Digital Unix file system is the main file system used. Open Genera includes an implementation of NFS that you can use to access UNIX files from the Genera environment. We strongly recommend you use NFS and the UNIX file system in working with the Open Genera. If you are unfamiliar with using file systems with Genera, see the section "Naming of Files".This section provides general information on the use of file systems with Genera.

Open Genera File System Access Through Symbolics NFS

Symbolics NFS is a user-transparent remote file access protocol. This system is a fully symmetrical implementation of the Sun Network File System protocol, built on top of the Sun Remote Procedure Call (RPC) protocol, which in turn is built on the Sun eXternal Data Representation (XDR) standard, all as specified in the Sun Network File System Protocol Specification, Revision B of 17 February 1986. The transport medium used is UDP/IP.

Symbolics NFS provides file transfer, direct file access, directory lookup, and file attribute modification. It allows the Open Genera environment to act as an NFS client. In addition, Symbolics NFS allows you to access files using the standard Genera activity interfaces (for example, you can use `c-X c-F` in Zmacs, or the Show Directory command from the Command Processor). Because the NFS protocol is UNIX oriented, there are some behavioral differences from the other remote file protocols available on Genera, primarily in status-line states and pathname syntax.

You can modify UNIX required characteristics with customization parameters — variables or user properties that can be assigned to namespace objects. See the section "User Properties for Configuring a UNIX File System".

Accessing an OSF/1 File System on Open Genera

Files on a UNIX file system can be accessed from Genera using pathnames that follow UNIX syntax conventions. All Genera file and directory commands will operate on the UNIX file system.

Typically, you access UNIX files using the format:

host:/*directory*/*filename*...

The word *host* represents the name of a UNIX host. *Directory* specifies the directory (or subdirectory) in which the file *file-name* resides. The slash character (/) separates pathname components. This format can be used for local or remote hosts.

UNIX Pathname Completion

Genera's completion facilities can access the file systems of various hosts, including Sun computers. To perform pathname completion, Genera looks in the host's file system and returns a new, possibly more specific string than the one that you have entered, expanding any unambiguous abbreviations in a host-dependent fashion.

Wildcard syntax is supported for UNIX pathnames. An asterisk (*) is used to specify a canonical type and is equivalent to the keyword argument **:wild**. One common use of the double asterisk (**) is to specify all directories and subdirectories; it is equivalent to the keyword argument **:wild-inferiors**. The double asterisk wildcard is supported only for UNIX files accessed by means of NFS.

Use the c-/ and c-? keystrokes to see multiple possibilities for what you have already typed. For more information about using these keystrokes, see the section "c-? and c-/".

UNIX Pathname Merging

Merging of UNIX pathnames is identical in most respects to pathname merging of LMFS pathnames. Like files in LMFS, UNIX pathnames are merged against the default. You need to specify only those fields (name, type, or version) that differ from the context-dependent default to create a new pathname. However, unlike LMFS pathnames, UNIX pathnames are case-sensitive. A complete discussion of Genera-style pathname merging appears in the section "Pathname Defaults and Merging".

The next examples show pathname merging when a new host or directory is specified:

Default	*Input*	*Merged*
ENIAC:/u0/mwra/foo.lisp	MARK-I:	MARK-I:/u0/mwra/foo.lisp

Default	*Input*	*Merged*
ENIAC:/u0/mwra/foo.lisp	/etc/	ENIAC:/etc/foo.lisp
ENIAC:/u0/mwra/foo.lisp	patches/	ENIAC:/u0/mwra/patches/foo.lisp
ENIAC:/u0/mwra/foo.lisp	../mbta/	ENIAC:/u0/mbta/foo.lisp

Since UNIX does not recognize that pathnames have a name or type, it is possible for UNIX filenames to have any number of dots in them. Because of this, names and types are not automatically defaulted during the merging process:

Default	*Input*	*Merged*
ENIAC:/u0/mwra/foo.lisp	bar	ENIAC:/u0/mwra/bar
ENIAC:/u0/mwra/foo.lisp	.bin	ENIAC:/u0/mwra/.bin
ENIAC:/u0/mwra/foo.lisp	...dots...	ENIAC:/u0/mwra/...dots...

However, names can be merged if you explicitly specify this in the input to be merged. You can indicate whether you want the name or type specified by using the double-arrow character "↔", entered with the SYMBOL-1 key combination:

Default	*Input*	*Merged*
ENIAC:/u0/mwra/foo.lisp	bar.↔	ENIAC:/u0/mwra/bar.lisp
ENIAC:/u0/mwra/foo.lisp	↔.bin	ENIAC:/u0/mwra/foo.bin

The double-arrow character (↔) indicates only whether an entire name or type component should be defaulted. If there are more characters in a name or type pathname component, it means nothing special:

Default	*Input*	*Merged*
ENIAC:/u0/mwra/foo.lisp	bar.b↔	ENIAC:/u0/mwra/bar.b↔
ENIAC:/u0/mwra/foo.lisp	x↔.bin	ENIAC:/u0/mwra/x↔.bin

If a filename has more than one dot in it, Genera interprets the last dot as the separator between the name and type component, and the rest of the dots as part of the name component. Therefore, using ↔ at the end of a filename will result in the type being defaulted, but using ↔ anywhere else in the filename has no effect:

Default	*Input*	*Merged*
ENIAC:/u0/mwra/foo.lisp	...dots...↔	ENIAC:/u0/mwra/...dots...lisp
ENIAC:/u0/mwra/foo.lisp	↔...dots...	ENIAC:/u0/mwra/↔...dots...

UNIX Backup Files

Through Symbolics NFS, Genera maintains backup file versions for UNIX using a numbering scheme that is compatible with those UNIX programs that do not recognize version numbers.

If you create a new file, foo.c, its creation is recorded as:

```
foo.c       1:00pm
```

When a new version of the file is written, the previous version is assigned a backup version number:

```
foo.c       1:14pm
foo.c.~1~   1:00pm
```

This process continues as you write other new versions of a file:

```
foo.c       2:03pm
foo.c.~2~   1:14pm
foo.c.~1~   1:00pm
```

You can set the number of backup versions that are retained by Genera with the "NFS-GENERATION-RETENTION-COUNT User Property" for host namespace objects.

Files with backup version numbers are not normally represented internally to Genera with pathname version numbers. Consequently, no Genera utility treats backup file copies as files with versions. For example, Dired and the Clean File command do not behave as if file backup copies are older versions of the same file.

Using SCT with a UNIX File System

The System Construction Tool (SCT) presents a special case in using UNIX backup files. SCT requires a version number for the files it uses. For directories maintained by SCT, files are maintained so that the newest versions of the files have an explicit version number. (See the section "System Construction Tool" for background information on SCT).

Use the name of a UNIX directory to specify that files contained in it are to be maintained by SCT and should all end with explicit version numbers. If the name of a UNIX directory ends with the `.sct` file extension, Symbolics NFS creates all new files under that directory with explicit version numbers.

Here's an example. Suppose a `sys.translations` file looked like this:

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER -*-

(fs:set-logical-pathname-host "SYS" :translations
  '(("**;" "ACME-UNIX:/usr/share/symbolics/rel-8-0/sys.sct/**/")))
```

In this case, all new files created with Symbolics NFS under `/usr/share/symbolics/rel-8-0/sys.sct/` are created with explicit version numbers in the newest versions of files, meeting SCT requirements. This is true for all subdirectories of a directory whose name ends in `.sct`.

The double-asterisk (**) wildcard can be used only in UNIX pathnames that are accessed by means of NFS. Other UNIX file protocols, such as FTP, do not recognize this wildcard. For further information on the use of the double-asterisk wildcard, see the section "LMFS Pathnames".

Genera considers UNIX pathnames with any directory components ending in the string ".sct" to be for SCT-maintained files, and hence to contain version numbers. The syntax that Genera uses for backup file version numbers in UNIX pathnames is a number surrounded by tildes separated by a dot from the file type.

This pathname is considered to have a version number:

```
ENIAC:/u0/mwra/visi-brain.sct/foo.lisp.~23~
```

So merging with it as a default behaves like this:

Default	*Input*	*Merged*
`ENIAC:/u0/dpw.sct/code/foo.lisp.~23~`	`bar.ₓ`	`ENIAC:/u0/dpw.sct/code/bar.lisp`
`ENIAC:/u0/dpw.sct/code/foo.lisp.~23~`	`ₓ.bin`	`ENIAC:/u0/dpw.sct/code/foo.bin`

Note that backup file version numbers are are not defaulted as part of the merging operation.

You can set the number of backup versions that are retained by Genera with the "NFS-GENERATION-RETENTION-COUNT User Property"for host namespace objects.

If you specify a pathname using the literal string "HOST" as the name of the UNIX host, Open GEnera will interpret the pathname as the name of a file on the DEC Alpha workstation running Open Genera.

User Properties for Configuring a UNIX File System

NFS-REMOTE-FILESYSTEM User Property

Syntax: NFS-REMOTE-FILESYSTEM *filename qualified-filename*

A user property for host namespace objects. The *filename* argument represents a pathname that specifies a directory on the host. The *qualified-filename* argument specifies a device. A device is defined as a directory on another host that is mounted on *filename*, a directory on the local host.

If the mount points for a UNIX host are unknown when Symbolics NFS accesses a file, it assumes that the file is local to that UNIX host. If the file is on some other host, in a file system that was NFS mounted by the UNIX host, Symbolics NFS can determine where the file really is only in cases where Symbolics NFS knows that the file system has been mounted by the UNIX host.

NFS-TRANSFER-SIZE User Property

Syntax: NFS-TRANSFER-SIZE *integer*

A user property for host namespace objects. It specifies the number of bytes of data requested or sent by each network RPC call when transferring whole files to

and from this host. When this property is present on the host namespace object, the value overrides the relevant default.

NFS-AUTOMOUNT User Property

Syntax: NFS-AUTOMOUNT *user-property*

A user property for host namespace objects. It informs the Symbolics client that an NFS server is running the automount program so Symbolics NFS can duplicate the file system layout of the NFS server. The value of the *user-property* argument should be the arguments given to the automount program running on the NFS server. See the UNIX man page for the automount program for more information about the automount program.

If the value of this user property is a single token that does not begin with the character N, Symbolics NFS assumes the host is running the automount program with no arguments. Symbolics NFS then consults the auto.master NIS map for automount configuration information.

If the value of the user property is more than one token, the value is interpreted as the arguments to the automount program running on the server.

NFS-GENERATION-RETENTION-COUNT User Property

Syntax: NFS-GENERATION-RETENTION-COUNT *integer*

A user property for host namespace objects. Sets the number of file versions retained by NFS when superseding an existing file. The default is 5.

Open Genera Network Services

Open Genera shares the network interface of its host, and appears on the host's network just like any other remote system. Although the host workstation and Open Genera share Ethernet hardware addresses, they use separate *protocol*, or software addresses, and therefore appear as separate hosts on the network. Other hosts request network services from either operating system independently, using the appropriate protocol addresses.

The Open Genera SEE offers extensive networking facilities and adapts easily to heterogeneous, multivendor environments. It includes protocols and services to support many different operating systems, and automatically chooses the best protocols, hosts, and routes to obtain a requested service. For a summary of the principal network services supported by Open Genera,see the section "Summary of Open Genera Network Services".

The Open Genera software distribution includes the Symbolics IP/TCP network product, to support communication with the host workstation. Unlike other Symbolics systems, which generally rely only on the Chaosnet protocol, Open Genera uses both Chaosnet and IP/TCP protocols, and uses IP/TCP protocols exclusively to communicate with its host.

In a network site containing a number of UNIX workstations and Symbolics workstations, the network is described by several different databases:

The Domain Name system
: The Domain Name system provides a consistent naming scheme for the network objects (typically just computers and organizations) on the Internet.

The Symbolics Namespace system
: The Symbolics Namespace system describes each networked entity; entities include users and shared peripherals in addition to computers. In particular, it describes the network services and protocols supported by each host.

UNIX NIS
: The Digital Unix NIS network database provides information needed to communicate with UNIX systems, such as user ids, group ids, and passwords.

The Open Genera distribution includes support for the Internet Domain system and UNIX NIS.See the section "UNIX UID and GID Mapping".

Summary of Open Genera Network Services

Genera supports a wide variety of network services, using a variety of network protocols. This section describes the principal network services supported by Open Genera. For related information: see the section "Generic Network Services".

FILE Service

FILE service gives Genera access to files stored on another computer. Open Genera supports file access using the Symbolics NFILE or QFILE protocols, the Sun NFS protocol, or the Internet FTP protocol. Note that Open Genera does not itself offer file service as it has no filesystems of its own.

LOGIN Service

LOGIN service allows remote hosts to log in to Genera and interact with its command processor, and vice versa. Open Genera software supports remote login using the Symbolics 3600-LOGIN protocol, or the Internet TELNET and SUPDUP protocols. See the section "Using the Terminal Program".

RPC Service

RPC service allows Genera to invoke procedures on a remote host using the Sun RPC protocol, and vice versa.

UNIX-REXEC Service

UNIX-REXEC service allows Genera to execute commands on remote UNIX hosts using the UNIX REXEC protocol, and vice versa. See the section "Remote Command Execution".

X-WINDOW-SYSTEM Service

X-WINDOW-SYSTEM service allows Genera to present its user interface on the console of a remote host. Open Genera supports X-WINDOW-SYSTEM service as a client only; it does not include an X server. Open Genera supports version 11 of the X protocol.

SHOW-USERS Service

SHOW-USERS service allows Genera to determine what users are logged in to a remote host, and vice versa. Open Genera supports SHOW-USERS service using the Symbolics NAME protocol, or the Internet ASCII-NAME protocol. See the section "Show Users Command".

STORE-AND-FORWARD-MAIL Service -- not supported

STORE-AND-FORWARD-MAIL service allows Genera to deliver mail messages to a user on a remote host. Open Genera supports STORE-AND-FORWARD-MAIL service using the Internet SMTP protocol. Open Genera itself does not include facilities to receive mail messages from a remote host.

SEND Service

SEND service allows Genera to deliver interactive messages to a user on a remote host. Open Genera supports SEND service using the Internet SMTP protocol and the UNIX TALK protocol.

HARDCOPY Service

HARDCOPY service allows Genera to print text and/or graphics on a printer attached to a remote host. Open Genera supports HARDCOPY service using the Symbolics LGP and PRINTER-QUEUE protocols, or the UNIX LPD protocol. See the section "How to Get Output to a Printer".

Remote Command Execution

The UNIX-REXEC network service allows a host to invoke the command interpreter of another host. The Execute Command command provides the Genera user with a convenient interface to this facility. The **tcp::make-rexec-stream** function allows Genera programs to invoke this service and interact with a remote program directly.

Execute Command Command

Execute Command *host command* &key *source*

Passes the specified command to the specified host, to be executed by its command interpreter.

host	The name of a host running UNIX or Genera.
command	A string to be interpreted by that host's command processor. This must usually be enclosed in double quotes.
:Source	An optional source of input to the command, which may be a file, editor buffer, the name of a Genera stream, or the output of a Genera command.

For example, to see all the UNIX processes on host ENIAC that are executing some part of X11, type:

```
Command: Execute Command (on host) ENIAC (command) "ps -aux | grep X11"
Password for logging in to ENIAC as tsw (or Escape to change user id):
root    159  0.0  0.0  120    0 ?  IW  Aug 18  0:00 /usr/bin/X11/xdm
root    162  0.0  8.6 2656  624 ?  S   Aug 18530:08 /usr/bin/X11/X :0 -fp /u
root    161  0.0  0.0  120    0 ?  IW  Aug 18  0:00 /usr/bin/X11/xdm
root    163  0.0  0.0  168    0 ?  IW  Aug 18  0:00 /usr/bin/X11/xdm
tsw     815  0.0  2.6   40  192 ?  S   12:01   0:00 grep X11
```

tcp::make-rexec-stream *host command* &key *user-name password* *Function*

Uses UNIX-REXEC to execute the specified command on the specified remote host, providing a stream connected to that command's standard input and output streams.

To simply execute a command and see the resulting output, use the Execute Command command.

This example uses **tcp::make-rexec-stream** and a UNIX C shell interpreter to kill all the processes running the `/etc/rmt` server.

```
(cp:define-command (com-kill-rmt-processes :name "Kill RMT Processes"
                                           :command-table "User")
    ((host 'net:host))
  (with-open-stream (stream (tcp::make-rexec-stream host "csh"))
    (write-string "ps -ax | grep rmt | grep -v grep; echo end" stream)
    (terpri stream)
    (force-output stream)
    (loop as string = (read-line stream nil)
          while (not (null string))
          as id = (parse-integer string :junk-allowed t)
          while (integerp id)
          do (format stream "kill -1 ~D~%" id))))
```

This example uses **tcp::make-rexec-stream** to connect a UNIX C shell to Genera's ambient ***terminal-io***. It uses a background process to continuously copy UNIX output to ***standard-output***, a useful technique. To terminate this example, either abort using c-ABORT, log out of the UNIX shell, which will close the connection, or enter an EOF using the FUNCTION END key combination.

```
(cp:define-command (com-toy-unix-shell :name "Toy UNIX Shell"
                                       :command-table "User")
    ((host 'net:host))
   (with-open-stream (stream (tcp::make-rexec-stream host "csh"))
     (let ((process nil))
       (unwind-protect
           (progn
             (setq process (process-run-function "Rexec source"
                             #'(lambda (input output)
                                 (stream-copy-until-eof input output)
                                 (setq process nil))
                             stream (cli::follow-synonym-stream *standard-output*)))
             (loop as string = (read-line *standard-input* nil)
                   while (not (null string))
                   do
               (format stream "~A~%" string)
               (force-output stream)))
         (when process
           (process:kill process))))))
```

System Conditionalization on Open Genera

It is occasionally necessary for Genera programs to distinguish between the different platforms Genera is supported on. Programs might need to be conditionalized between the 3600 and Ivory architectures, since Ivory supports some software packages and architectural features not available on the 3600 series of machines. In general, Ivory and Open Genera world-loads and compiled files are not binary compatible. Such conditionalization should be done at read time (that is, compile time), using the **#+3600**, **#+IMach**, and **#+VLM** reader macros. Use **#+IMach** to conditionalize for both Ivory-based machines and VLM, use **#+VLM** to conditionalize for VLM only, and use **#+(and IMach (not VLM))** to conditionalize for Ivory-based machines only.

Programs compiled for either the 3600 or Ivory architectures, and world loads containing those programs, should work on any product using their target architecture. Therefore, conditionalization between product family members cannot be done at read time (that is, at compile time); that's why **#+MacIvory** is not supported, for example.

The proper way to distinguish between product family members is to use the **sys:system-case** macro. Open Genera is designated by the **VLM:system-case** keyword, which means precisely that Genera is running on a DEC Alpha. The **sys:system-case** macro should be used only as a last resort; often more precise

conditionalization techniques, such as direct queries about the size of the screen or the availability of an I/O service, are available.