Genera 7.2
Patch Notes

symbolics™

Cambridge, Massachusetts

Genera 7.2 Patch Notes
 # 999059

February 1988

This document corresponds to Genera 7.2 and later releases.

Symbolics, Inc.
11 Cambridge Center
Cambridge, MA 02142

Text written and produced on Symbolics 3600™-family computers by the Documentation Group of Symbolics, Inc.
Text masters produced on Symbolics 3600™-family computers using Symbolics Concordia™, a system for supporting document development, and printed on Symbolics LGP2 Laser Graphics Printers.

Printed in the United States of America.

Printing year and number: 90 89 88 5 4 3 2 1

Table of Contents

List of Figures

1. Genera 7.2: Introduction and Highlights

These notes accompany the release of Genera 7.2. They describe changes made since Genera 7.1.

A complete list of changes appears in the Table of Contents. The notes cover the following topics:

Overview of Genera 7.2

This section discusses general issues connected with Genera 7.2:

- Support for new hardware
- World size
- Compatibility with previous releases
- Availability of sources
- The sys:unsupported; directory

Changes to the Lisp Language in Genera 7.2

This section describes changes relevant to the Lisp language.

Changes to Flavors in Genera 7.2

This section describes changes to Flavors in Genera 7.2.

Changes to Zmacs in Genera 7.2

This section describes changes in the Zmacs editor. The most important changes are:

- The addition of a complete Undo facility.
- The ability to run multiple Zmacs processes, each with its own state.
- New documentation of indentation.
- Support for non-Symbolics character sets.

Changes to Zmail in Genera 7.2

This section describes changes in Zmail. Changes include:

- Zmail now has control-X commands, such as c-X B for selecting a sequence and c-X c-F for reading in a mail file.

- c-m-Y and c-m-sh-Y now work in Zmail.

- Zmail now queries before executing time-consuming or "dangerous" commands like Save or Expunge. A new profile option provides control over this feature.

Changes to Utilities in Genera 7.2

This section describes changes in what would be called the operating system and utilities on other more conventional computers. This includes the Debugger, the garbage collector, and various system keyboard features. The most important changes are the following:

- A Metering Interface is now available, enabling you to measure various aspects of performance of programs, including speed, page faults, and consing. The Metering Interface also supports Call Tree metering, which gives you information on the calling behavior of functions.

- The reliability and performance of the garbage collector have been improved significantly.

- Dynamic GC flipping has been sped up by a factor of 50 to 100.

- To help the garbage collector reclaim more memory, you can now specify a GC cleanup to precede dynamic or immediate garbage collection.

- Document Examiner now uses Dynamic Windows and offers an improved topic search capability. The commands in Document Examiner's command pane have been renamed for easier use.

- Genera 7.2 introduces **sct:define-distribution-system**, which is the reimplementation of the Release 6 concept of a "distribution pseudo-system". This allows you to specify a "system" for distribution that is an arbitrary collection of files and may have wildcard pathnames in it.

- A new Display Debugger and a new Font Editor have been added in Genera 7.2. Both use Dynamic Windows technology, and replace the old Window Debugger and Font Editor, which remain separately loadable.

- The Namespace Editor now uses Dynamic Windows, AVV menus, and more informative prompts. It also performs extensive error checking to ensure that the user makes valid entries.

- The *Printer Installation Guide* is now loaded as part of the Print system.

Changes to the User Interface in Genera 7.2

This section describes changes to the user interface. Changes include:

- In Genera 7.2, more system activities (notably Document Examiner, the new Display Debugger, the Terminal program, and the namespace editor) use Dynamic Windows.

- A new facility, the Select Key Selector, lets users assign activities to their choice of SELECT key combinations.

- Help text for SELECT HELP, FUNCTION HELP, SYMBOL-HELP, and Debugger c-HELP is now printed in Dynamic Windows that have a uniform interface for scrolling, searching, and exiting. In addition, the Notifications window is now dynamic.

- The Notifications window is now dynamic.

- A new facility, HackSaws, provides random useful bits of folklore and other information about Genera that you can access from most system activities.

Changes to the User Interface Programming Facilities in Genera 7.2

This section describes changes to the facilities available to application programmers for building user interfaces. The changes include the following:

- Dynamic Windows output performance has improved substantially.

- Many bugs in Dynamic Windows mouse sensitivity have been fixed, and performance of mouse sensitivity has been greatly improved.

- The presentation system is much faster and more reliable than in Genera 7.1.

- User interface programming facilities documentation includes much new conceptual information and many new examples. Documentation is also provided for new and previously undocumented old user interface programming facilities functions.

- The graphics substrate has been rewritten for 7.2. It is almost completely compatible with previous versions. The new substrate is generic, which means that there is a uniform interface for accessing the different facilities provided by different graphical output devices.

Changes to the File and Streams in Genera 7.2

This section describes changes to files and streams.

Changes to Networks in Genera 7.2

This section describes changes in network implementation, interface, and protocols. Changes include:

- Performance and reliability of synchronous and asynchronous serial streams has been improved in 7.2 for all 3600-series machines.

- 3620s, 3630s, and 3650s can now be used as gateways over synchronous serial lines for Chaos and IP.

- Genera 7.2 includes support for non-multi-homed networks in a multi-namespace environment.

Changes to the FEP in Genera 7.2

Several new FEP commands have been added to support the new netbooting and autobooting features. All FEP commands are now documented.

Changes to Tape and Disks in Genera 7.2

This section describes new tape and disk features.

Genera 7.2: Operations and Site Management

This section describes changes to the system and site configuration features of the system. These changes are important to the people who are responsible for site installation and maintenance.

- Netbooting has been added. This is the ability to boot a world from a remote machine and run it without having the booted world-load file on the local machine.

- Autobooting is available on machines with the G208 FEP EPROMs. Systems with autobooting enabled reboot automatically, without attention, after any FEP reset, such as a power-up event.

- Genera 7.2 contains new tools to automate site installation and configuration.

Changes to Documentation in Genera 7.2

This section describes changes to the printed and online documentation.

- Books whose contents have been changed substantially for Genera 7.2 have been republished.

- The *User's Guide to Symbolics Computers* has been expanded into four guides offering different approaches to learning how to use Genera.

- Books 7A and 7B of the documentation set have been rewritten to follow the model used in Books 2A and 2B — conceptual information is in volume A, with volume B a dictionary of functions.

Notes and Clarifications for Genera 7.2

This section contains explanations and clarifications of items that people found confusing in previous releases and documentation.

2. Overview of Genera 7.2

2.1. New Hardware Supported by Genera 7.2

Genera 7.2 supports the following new hardware:

- 3630 and 3653 systems
- Color consoles
- Maxtor XT4380 disks
- 6250 bpi tape drive

2.2. Compatibility of Genera 7.2 with Previous Releases

All programs compiled in Genera 7.1 will load and run properly in Genera 7.2, unless they use certain undocumented internal interfaces that very few programs use. Programs compiled in Genera 7.2 will generally not run properly when loaded in Genera 7.1. Files of data produced by **sys:dump-forms-to-file** in Genera 7.2 will load correctly into Genera 7.1, and vice versa.

2.2.1. Compatibility Terminology

In a release version number, such as "Genera 7.2", the first number ("7") is the major version number, while the second number ("2") is the minor version number. In a minor release, the minor version number is incremented while the major version number remains unchanged. In a major release, the major version number is incremented and the minor version number is reset to zero.

"Upward compatibility" refers to the ability to take data or a program generated in an earlier release and use it in a later release.

"Downward compatibility" refers to the ability to take data or a program generated in a later release and use it in an earlier release.

2.2.2. Upward and Downward Compatibility in Genera 7.2

Symbolics' general inter-release compatibility policy differentiates between program compatibility and data compatibility. In addition, it differentiates between major releases and minor releases.

Upward compatibility for data files is always provided. This includes text files, BIN files generated by **sys:dump-forms-to-file** that do not contain any compiled

functions, KBIN files, and so on. Downward compatibility for data files is generally provided, as long as the data file does not use any features that were not present in the older release. For example, text files that do not use characters of non-default style and do not contain any characters in non-standard character sets are downward compatible to older releases than text files that do contain such characters. KBIN files are an exception to the rule of downward compatibility, as their format is too complex for downward compatibility to be practical.

Less compatibility is provided for compiled programs than for data, on the grounds that programs can be recompiled a lot more easily than data can be recreated.

Upward compatibility for compiled programs (in BIN files) is provided for minor releases. In rare cases, a documented feature or a feature that is known to be used by customers might be changed incompatibly in a minor release. This is done only when unavoidable and is always documented in the release notes. We try not to make incompatible changes to undocumented internal interfaces that we know customers use, but if we aren't aware that a customer is using an interface, and the interface is not documented, it's possible that we may change it incompatibly in a minor release.

Upward compatibility for compiled programs is not provided for major releases. The representation of compiled programs or the expansion of system macros is often changed incompatibly at a major release in order to provide higher performance or new features.

Downward compatibility for compiled programs is never provided, even in minor releases. The primary reason for this is that programs are very complex entities and it would be difficult to develop the system in such a way as to maintain such compatibility. More importantly, it would be totally impractical to test such compatibility during the quality assurance process, because of the enormous number of possible compatibility issues. Thus even if we claimed to provide downward compatibility for compiled programs, it would probably not work very well.

Upward compatibility for source programs is always provided for minor releases and usually provided for major releases. In a major release there may be some source changes necessary if an interface is changed, but this is always documented and we attempt to minimize it. Of course, source program compatibility might not be provided for programs that call undocumented internal interfaces. Downward compatibility for source programs is generally also available, especially across minor releases, provided the program has not been changed to exploit new features not present in the older release.

2.2.3. Compatibility Considerations in Genera 7.0

Genera 7.0 offered an unusually small amount of upward compatibility with the previous release. There were two reasons for this.

1. A change of language, from Zetalisp to Common Lisp, entailed changes to the representations of arrays and of characters. The change to characters, in particular, necessitated broad incompatible changes throughout Genera. This incompatibility was considered worthwhile because it made possible the adoption of the widely accepted Common Lisp language standard.

2. A large number of pending incompatible changes were batched together and released in Genera 7.0 in order to deliver the incompatibilities all at once, so users would have to make major changes to their programs only once. Releasing these incompatible changes made possible the implementation of a large number of features and the cleaning up of many old maintenance problems.

Future major releases will offer a considerably greater degree of upward compatibility, as we do not contemplate changing language again, and we no longer have a long list of necessary incompatible changes awaiting an opportunity for release.

2.2.4. Compatibility Considerations in Genera 7.2

Response to customer feedback is a primary reason why files compiled in Genera 7.2 might not load into Genera 7.1. For example, customers have reported the following:

- The expansions of the **cp:define-command** macro appeared excessive, and that space was being wasted and paging performance negatively affected. Genera 7.2 introduces a number of runtime helper functions that can be shared among more than one command. These functions do not exist in Genera 7.1.

- Aborting during **define-presentation-type** destroyed things so badly that you had to reboot. The macroexpansion has been changed to protect against this, but it depends on functions that are not present in Genera 7.1.

In some cases, changes were made to undocumented parts of the system that we do not expect users to rely on. For example:

The Font Editor and Document Examiner programs have been converted to Common Lisp. As a result, the **fed** and **sage** packages now contain explicit external symbols and require use of a double colon to access their internal symbols. If you do not use symbols from **fed** or **sage**, this will not affect you. If your program loads without warning, you are properly using the few exported symbols that were intended as external interfaces all along. If you get a message from trying to use an internal symbol as an external, you should check that the function you are calling still works the way you require. Note that every effort has been made to keep the functional interfaces compatible, so you will probably not run into this problem, anyway.

Symbolics is committed to maintaining and improving the new user interface substrate introduced in Genera 7.0. However, in order to release Genera 7.2 on time, this substrate has not yet reached a degree of flexibility which supports all previous styles of interaction. For this reason, both the DIred and FSEdit capabilities are still provided in the absence of an integrated single interface that can be tailored either way.

Also, a large number of programs have been converted to use a new consistent user interface. In most cases, this conversion is upwardly compatible, as an interface similar to the previous one now allows for correct mouse sensitivity and gen-

eral scrolling. In a few instances, the new program represents a significant change from the old. So, for instance, in Genera 7.0 and subsequent releases we have provided a Lisp Listener that does not have a scrolling output history. In Genera 7.2, we have supplied the old Window Debugger, Namespace Editor, and Font Editor programs in addition to the new Dynamic versions. These old programs have been compiled and tested for this release. They have not been enhanced in any way, but rather they will run exactly as before.

The old Font Editor program can be loaded by giving the command:

```
Load System FED
```

The Window Debugger and Namespace Editor programs can be found in the `sys:unsupported;` directory. For more information: See the section "New Directory for Unsupported Software", page 12. Genera 7.2 features have been designed with compatibility considerations in mind. For example:

Genera 7.2 introduces a generalization of the **graphics:** drawing functions introduced in Genera 7.0. The enhanced functions allow for arbitrary scaling, rotating, thickness, and patterns just as other modern graphics standards, such as PostScript, do. In all cases save one (**graphics:draw-ellipse**), the calling sequence is an upward compatible extension of the old. In cases where some old arguments do not fit well into the new model, such as the **:draw-end-point** argument to **graphics:draw-line**, or the **:value** argument to **graphics:draw-point**, the old arguments are supported at runtime, and style checked at compile time, with explicit instructions on how to write them in terms of the new generalized forms.

Scan conversion has been made consistent among all new drawing functions, which in some cases has meant a change in the actual pixels affected; details aare given elsewhere. However, the **:draw-** messages to windows have not been changed in any way since Genera 7.0, except to fix some prominent bugs. **:draw-string** has been made consistent with Release 6 (this change was in Genera 7.1), **:draw-wide-curve** handles the **tv:alu-xor alu** better, with still always draws the same shape (even though this shape is not always the correct one). The bug where **:draw-line** clips incorrectly and alters the slope of the line has not been fixed, because we could not assess how many programs might be relying on it. In all cases, the corresponding problems have been fixed in the **graphics:draw-** functions. This has almost always required introducing parallel sets of methods and functions in the interest of compatibility.

2.3. The Size of the Genera 7.2 World

The Genera 7.2 distribution world is approximately 33,000 blocks, which is about 5,000 blocks smaller than the Genera 7.1 distribution world.

Nevertheless, Genera 7.2 introduces significant new functionality that would normally have increased the size noticeably. Symbolics realizes that world size presents difficulties for many customers. Therefore, we have made every effort to keep it under control, within the compatibility constraints of this minor release.

Here is some information that may help you understand why the world is as large as it is and the details of what we have done about it.

2.3.1. Contribution of Code to the Size of the World

If the Genera 7.2 distribution world had been constructed by Genera 7.1 standards it would be 43,000 pages. That is, the new tools and functionality supplied with this release could have represented an increase in world size of greater than 10 percent. This would have been consistent with past growth patterns.

The size of the actual Genera 7.2 world has been reduced by a redesign of most of the larger system data structures. At the same time, nearly full compatibility has been retained and no major functionality has been lost.

Actual compiled code makes up only 6000 pages (15 percent of the world). The number of applications does not necessarily determine the size of the world. You cannot simply count the number of entries displayed by SELECT HELP, mark off those that you do not use, or assume your own users do not use, and conclude that you could get them back *m/n* percent of the world by taking them out.

The amount contributed by various applications varies enormously. For instance, programs like FrameUp and the Flavor Examiner take up less than 1 percent of the code (not of the system total), the Font Editor consumes about 1 percent, and Zmacs and Zmail each about 8 percent, to say nothing of the variability in the size of program-related data structures.

Symbolics contemplated removing a few programs that we do not believe are used by the large majority of our customers. As it turns out, these programs contribute very little functionally and therefore contribute very little in size. Moreover, we might have been wrong in our guesses, so we did not remove them.

We are aware of problems with attempting to install layered products successfully. We did not wish to further increase this burden by making large and critical parts of the system optional until we could better deal with the general problem of installation and the further combinatorial complications that more complex installation would require.

In a couple of instances, reimplementations of certain programs have been made available that are not fully compatible with the old version. To maintain compatibility for users of these, we have not provided either version in the world by default. These removals from the world have not significantly reduced its size and were not motivated by size considerations.

2.3.2. Consumers of Space in the World

Having said that code makes up only about 15 percent of the world, let us take a closer look at what does consume space. We will do this by examining how much space would have been consumed by the Genera 7.2 system according to Genera 7.1 standards and in some instances how this has been reduced in the actual system.

- **Flavor system data structures** take 7500 pages (18 percent). These structures have been tuned in favor of execution time rather than space reduction. While other tradeoffs along this dimension might be possible, we did not feel they were warranted in a minor release without further investigation and also feedback from users.

- **Debugging information** would have occupied 5200 pages (13 percent) by Genera 7.1 standards. A small fraction of this information is necessary for correct compilation, but most of it is used only when the associated function is on the stack in the Debugger. By putting this information into a more compact form, it has been reduced by about 1200 pages with no loss of information or compatibility. Only the internal storage slot in the compiled code object is changed. The accessor function, **debugging-info**, converts from the compact format to the fully expanded, compatible one.

  An additional 400 pages has been saved by forgetting information used to record the macros and constants expanded during compilation. Since this information is used only by the who-calls database, and only when call tracing for the system itself is enabled, it is written to a binary file that is automatically loaded by **si:enable-who-calls-database** under this condition. The time to load the file does not contribute significantly to the time needed for the code analysis this already does.

  Thus, debugging information has gone from 5200 pages by Genera 7.1 standards to 3100 pages.

- **Symbols, symbol names, and package hash tables** would have occupied about 7500 pages (12 percent). A side-effect of the compact encoding of the debugging information decreases the number of valid symbols. Some 500 pages of symbols, 300 pages of symbol names, and about 700 pages of packages have been reclaimed, reducing the pages occupied in Genera 7.2 to 3000 pages.

- The **documentation database** would have occupied 4000 pages (10 percent) by Genera 7.1 standards. This is because it contains complete information except for the actual contents of the documentation, including keyword and oneliner data. By compressing the database to include only the data necessary for topic name completion and candidate listing, this has been reduced to about 1500 pages in Genera 7.2

- **Pathnames, source file names, binary file attribute lists, and other artifacts of the system** take about 3000 pages (7 percent). This information remains in the Genera 7.2 system.

- **Completion tables for flavor and generic function names** would have occupied 800 pages (2 percent) by Genera 7.1 standards. By implementing package-sensitive completion, wherein the tables store only the name, we have halved the number of table entries, and eliminated virtually all strings (as they

are now all shared with symbol names). This works out to about 50 pages used. Needless to say, this change also fixes an important bug in the flavor-related commands.

- Error tables for the various microcode versions for each of the possible hardware configurations occupy 500 pages (1 percent) of the world. Since many problems can occur while installing a distribution world on a new machine, we did not remove the tables.

- **Mouse handler tables, bit arrays for windows, fonts, and windows themselves** occupy several hundred pages (about 1 percent each).

This accounts for about 80 percent of the total size of the world. We have attempted to avoid double counting anything. The remaining 20 percent is miscellaneous data structures for all the various programs not otherwise accounted for. There are no outstanding consumers, except perhaps the editor.

In addition to the above specific breakdown, one additional thing worth specifically noting is the space occupied by patches. In previous releases, patch file comments and patch file pathnames accounted for a large number of pages. Near the end of the release, there is always a flurry of patches. Each one of these patches adds another source file to the system, bloating the source file information mentioned above. Lately, many patch comments are fat strings, meaning one word per character. In a typical development world, these account for 900 pages alone. There are no patch files or patch comments in the Genera 7.2 distribution world. All patches have been loaded into the distribution world.

A good accounting of the changes since the last official release is given in the *Genera 7.2 Patch Notes*. In the world itself, only the base number for the world as shipped is recorded, so that patches supplied by the Software ECO service can still be loaded successfully.

2.3.3. Clearing Space After Installation

You may ask, "Is it possible to eliminate unwanted parts of the system after installing the basic distribution world?" Symbolics answers, "No, you can't." The system is highly integrated and therefore this maneuver is impractical for code and application-specific data structures.

2.4. More Sources Available in Genera 7.2

As of Genera 7.2, all system sources previously classified as Optional or Restricted are now Optional, and are available for purchase on a single tape. For more information, contact your sales representative.

2.5. New Directory for Unsupported Software

On your Genera 7.2 tapes, you'll find a directory called SYS:UNSUPPORTED;.

A large number of programs have been converted to use a new consistent user interface with dynamic windows. In most cases, this conversion is upwardly compatible, as an interface similar to the previous one now allows for correct mouse sensitivity and general scrolling. The new Document Examiner interface is an example of this.

In a few instances, the new program represents a significant change from the old. In those cases, we have attempted to preserve the older style for the convenience of users who rely on it, or simply are accustomed to it and do not wish to change. For instance, we continue to provide a "static" Lisp Listener that does not have a scrolling output history.

In Genera 7.2, there are new dynamic-window versions of the Font Editor, the Window Debugger and the Namespace Editor, but Symbolics continues to supply the old versions of these programs. The latter two programs are to be found in SYS:UNSUPPORTED;. (The old Font Editor remains in SYS:I01;.) These old programs have been compiled and tested for this release. They have not been enhanced in any way, but rather they will run exactly as before. No support is available and bugs will probably not be fixed.

Symbolics cannot guarantee that these programs will continue to run, or even be available, in all future releases.

3. Changes to the Lisp Language in Genera 7.2

3.1. Improvements to Lisp in Genera 7.2

3.1.1. New Array Function in Genera 7.2

Here is the argument list and brief description of the new array function in Genera 7.2:

sys:array-element-byte-size *array*
Given an array, returns the number of bits that fit into an element of that array.

3.1.2. New Bit-Vector Functions in Genera 7.2

Here are the new bit-vector functions in Genera 7.2:

bit-vector-cardinality *bit-vector* &key *(:start* **0***) :end*
Tests how many of the bits in the range are one's and returns the number found.

bit-vector-disjoint-p *bit-vector-1 bit-vector-2* &key *(:start1* **0***) :end1 (:start2* **0***) :end2*
Tests if two bit vectors are disjoint in a range specified by *:start1 :end1 :start2 :end2.*

bit-vector-zero-p *bit-vector* &key *(:start* **0***) :end*
Tests whether the *bit vector* is a bit vector of zeros in a range specified by *:start* and *:end.*

bit-vector-subset-p *bit-vector-1 bit-vector-2* &key *(:start1* **0***) :end1 (:start2* **0***) :end2*
Tests if one bit-vector is a subset of another bit-vector in a range specified by *:start1 :end1 :start2 :end2.*

bit-vector-equal *bit-vector-1 bit-vector-2* &key *(:start1* **0***) :end1 (:start2* **0***) :end2*
Tests if two bit vectors are equal in a range specified by *:start1 :end1 :start2 :end2.*

bit-vector-position *bit bit-vector* &key *(:start* **0***) :end*
If *bit-vector* contains an element satisfying *bit*, returns the index within the bit vector of the leftmost such element as a non-negative integer; otherwise **nil** is returned.

3.1.3. New Number Constants in Genera 7.2

Here is a list of new number constants in Genera 7.2:

least-positive-normalized-single-float
The value of **least-positive-normalized-single-float** is that normalized positive floating-point number in single-float format closest in value (but not equal to) zero.

least-negative-normalized-single-float
The value of **least-negative-normalized-single-float** is that normalized negative floating-point number in single-float format which is closest in value (but not equal to) zero.

least-positive-normalized-double-float
The value of **least-positive-normalized-double-float** is that normalized positive floating-point number in double-float format closest in value (but not equal to) zero.

least-negative-normalized-double-float
The value of **least-negative-normalized-double-float** is that normalized negative floating-point number in double-float format which is closest in value (but not equal to) zero.

least-positive-normalized-short-float
The value of **least-positive-normalized-short-float** is that normalized positive floating-point number in short-float format closest in value (but not equal to) zero.

least-negative-normalized-short-float
The value of **least-negative-normalized-short-float** is that normalized negative floating-point number in short-float format which is closest in value (but not equal to) zero.

least-positive-normalized-long-float
The value of **least-positive-normalized-long-float** is that normalized positive floating-point number in long-float format closest in value (but not equal to) zero.

least-negative-normalized-long-float
The value of **least-negative-normalized-long-float** is that normalized negative floating-point number in long-float format which is closest in value (but not equal to) zero.

3.1.4. New Number Functions in Genera 7.2

Here are the argument lists and brief descriptions of the new number functions in Genera 7.2:

random-normal &optional *(mean* **0.0***) (standard-deviation* **1.0***) (state* ***random-state****)*
Generates random numbers from the normal (Gaussian) distribution with mean *mean* and standard deviation *standard-deviation.*

print-exact-float-value
When this variable is set to **t**, it prints the exact number represented by a floating-point number, not the rounded verson which is normally printed by the printer.

3.1.5. New String Function in Genera 7.2

Here is the argument list and brief description of the new function **string-thin**:

string-thin *string* &key *(:start* **0***) :end (:remove-style* **t***) :remove-bits :error-if :area*
Returns a substring of *string* with the specified character-style information and bits removed.

3.1.6. New Table Functions in Genera 7.2

Here are the new table management functions in Genera 7.2:

sys:page-in-table *table* &key *:type :hang-p*
Brings back into main memory any swapped pages in *table* that have been swapped out to disk.

sys:page-out-table *table* &key *:write-modified :reuse*
Takes out to main memory any swapped pages in *table.*

sys:compile-table-flavor &rest *options* &allow-other-keys
Pre-compiles a flavor table. Put this function into one of your system files to pre-compile flavor tables, thus saving you some compile time.

sys:with-table-locked *(table)* &body *body*
Locks a table around code.

3.1.7. New Resource Conditions in Genera 7.2

There are three new resource conditions in Genera 7.2. These resource conditions are documented in "Resource Errors Based on si:resource-error".

See the flavor **si:resource-error** in *Symbolics Common Lisp -- Language Concepts.* See the flavor **si:resource-extra-deallocation** in *Symbolics Common Lisp -- Language Concepts.* See the flavor **si:resource-object-not-found** in *Symbolics Common Lisp -- Language Concepts.*

3.1.8. New defstruct Option

:inline is a new option for **defstruct** and **zl:defstruct.**

:inline Values can be **:accessors, :constructor, :copier, :predicate,** or the name of a slot. Defaults to the compiling accessors, constructors, and predicates inline. Note that the default is for most functions to be compiled inline. For example:

```
(:inline :constructor x-pos y-pos)
```

This example causes the constructor functions, spaceship-x-pos, and spaceship-y-pos, to be compiled inline. For information on inline functions: See the section "Inline Functions".

3.1.9. dotimes and zl:dotimes Fixed

dotimes and **zl:dotimes** have been fixed to compile properly when using the iteration variables **ignore** and **ignored**.

3.1.10. Improvements to Characters in Genera 7.2

Encryption and decryption is fixed to work in Zwei buffers and Zmail messages containing characters which have non-**nil** character styles.

3.1.11. Improvements to List Function in Genera 7.2

The function **list** is fixed to work properly when called with a large number of arguments, from interpreted code.

Previously, this error message appeared when you used **list** with a large number of arguments:

```
Trap: Attempt to make a stack frame larger than 256. words
```

Some special forms may still generate this error.

3.1.12. Improvement to Number Function in Genera 7.2

%logldb now loads out of bignums, allowing a byte size of up to 32 bits, including the sign bit.

3.1.13. Improvements to Sequence Functions in Genera 7.2

In Genera 7.2, we have fixed a bug in the function **substitute-if-not** Previously, the function gave incorrect results for vectors.

The run-time speed of Common Lisp sequence functions is also improved.

3.1.14. Improvements to String Functions in Genera 7.2

The following string functions have had their declared return-values changed in Genera 7.2:

read-from-string now has the declared return-value: (form index)

accept-from-string now has the declared return-value: (object type index)

3.1.15. Improvements to the Reader Macros #+ and #-

The reader macros #+ and #- have been fixed so that they do not signal package errors when used on constructs such as:

```
#+system::cmu
```

and

```
#+cl:alcatraz
```

3.1.16. Genera 7.2 Allows Printing of TINY Character Styles

A character style is a combination of three characteristics that describe how a character appears: family, face, and size. Prior to Genera 7.2, the LGP2 did not let you print anything that contained TINY character styles for the FIX character style family. This has been fixed in Genera 7.2.

3.2. Incompatible Changes to Lisp in Genera 7.2

3.2.1. Changes in String Functions

The following functions now signal an error if the argument *string* is anything but a string. Previously, when passed a non-string argument, these functions tried to coerce the argument into a string.

zl:string-capitalize-words
zl:string-upcase
zl:string-downcase
zl:string-flipcase
zl:string-nreverse

The following functions do not signal an error unless the *copy-p* argument is specified **nil**.

nstring-upcase
nstring-downcase
nstring-flipcase
nstring-capitalize
nstring-capitalize-words
string-nreverse

3.2.2. make-hash-table Incompatibility

In Genera 7.1 you could make a hash table with = as the **:test**, and it defaulted the hash function to **cli::xequal-hash**.

An incompatibility arises if one of your programs uses the following construct without specifying the keyword **:hash-function**:

```
(make-hash-table :test '= ...)
```

Genera 7.2 requires that you specify a **:hash-function**.

Why Doesn't Symbolics Common Lisp Support = in Hash Tables?

In the following examples, note the differences in the two hash codes, even though the two numbers are =:

```
(cli::xequal-hash 1.4e20)
=> 1626528982

(cli::xequal-hash 14000000000000000000)
=> 179306497


(equal 1.4e20 14000000000000000000)
=> NIL

(= 1.4e20 14000000000000000000)
=> T
```

The differences are because **cli::xequal-hash** corresponds to **zl:equal**, not =, and Common Lisp dictates slightly different rules for these two equality functions.

You might think that a possible solution is to substitute a different hash function, one specialized for =. This is not possible, however, since Common Lisp declares that the = function is not transitive:

```
(= 1.4e20 14000000000000000000)
=>  T

(= 1.4e20 14000000000000000001)
=>  T

(= 14000000000000000000 14000000000000000001)
=> NIL
```

This means that the hash function for = , when applied to an integer, has to ignore all digits past the ones corresponding to the precision of the smallest floating-point representation. This makes it a poor hash function for tables that only use integers for keys. For this reason, Symbolics Common Lisp does not support = in hash tables.

Workaround:

The following workaround, which is not semantically correct, gives the wrong answers if you use floating point numbers. Customers with old versions of ILA NFS can use this workaround to make old versions not signal an error, since NFS does not use floating point numbers:

```
(cli::add-test-function-hash-function '= 'cli::xequal-hash)
```

3.2.3. Hash Tables and :test

In Genera 7.1 and earlier releases, when creating a hash table with **make-hash-table**, you could use any function as the **:test** function of the hash table. The default hash function in Genera 7.1 was the one for **zl:equal**, which did not work correctly for most other predicates. In Genera 7.2, if the **:test** function is not in the list below, you must explicitly provide a **:hash-function**.

:test Determines how keys are compared. Its argument can be any function; **eql** is the default. If you supply one of the following values or predicates the hash table facility automatically supplies a **:hash-function**: **eq** , **eql** , **equal** , **char-equal**, **char=**, **string-equal**, **string=**, **zl:equal**, **zl:string-equal**, **zl:string=**. If you supply a value or predicate that is not on this list, then you have to supply a **:hash-function** explicitly.

Note: You must supply a **:hash-function** if the test is =.

3.2.4. char and schar Functions Changed

char and **schar** are changed in Genera 7.2. Now their argument lists are:

string index

Previously their argument lists were:

array &rest *subscripts*

3.2.5. once-only Requires Keyword

once-only now requires the **&environment** keyword.

The argument list for **once-only** is:

((variable-name... *&environment* environment) &body body)

If you do not supply the **&environment** keyword, the following error message appears:

```
For Function FOO
  A call to ONCE-ONLY was found with no &ENVIRONMENT keyword.

  Local macros will not expand properly inside the body.
  Supply the lexical environment, derived from the &ENVIRONMENT
  parameter in a DEFMACRO, MACROLET, or DEFMACRO-IN-FLAVOR.
```

3.2.6. :allow-other-keys Is No Longer Valid

Functions with &key arugments now allow the :allow-other-keys keyword when its value is not **nil**. This is an example of a construct that used to work, but now signals an error:

```
((lambda (&key x) x) :allow-other-keys nil)
```

The error generated is:

```
(:ALLOW-OTHER-KEYS is not a valid keyword)
```

4. Changes to Flavors in Genera 7.2

4.1. New Features of Flavors in Genera 7.2

4.1.1. New Flavor Macro: flavor:with-instance-environment

Genera 7.2 makes the following macro available:

flavor:with-instance-environment *(instance env)* *Macro*
&body *body*

Within the *body*, the variable *env* will be bound to an interpreter environment for the specified *instance*. The primary use of this is to create a listener loop like that of the debugger when examining a method, in which you can reference an instance's instance variables and internal functions directly.

4.2. Improvements to Flavors in Genera 7.2

4.2.1. Flavor Constructors Can Use &allow-other-keys in 7.2

In 7.2, you can use the **&allow-other-keys** keyword in the lambda-lists of flavor constructors.

4.2.2. New :Using Instance Variables Option for Show Flavor Methods

Genera 7.2 adds a new keyword option for the Show Flavor Methods command:

:Using Instance Variables
Enables you to request only those methods that use the specified instance variables.

4.2.3. compile-flavor-methods Forms Can Be Interpreted in 7.2

Previous to 7.2, **compile-flavor-methods** forms could not be interpreted, only compiled. This restriction is removed in 7.2.

4.2.4. Changed Function Specs for Whoppers in 7.2

In 7.2, the function specs for whoppers are in this format:

(flavor:ncwhopper *generic-function flavor*)

Previously the function specs for whoppers were in this format:

(flavor:whopper *generic-function flavor*)

This is a compatible change; the old function specs for whoppers continue to work, so programs compiled in 7.0 or 7.1 will work in 7.2 without recompilation.

The motivation for this change was to reduce the number of combined methods in the Lisp world. The new type of whopper does not need a combined method to implement **continue-whopper**.

4.2.5. Clarifications to Flavors Documentation in 7.2

Clarifications have been added to the documentation of some **defflavor** options: See the section "Complete Options for **defflavor**" in *Symbolics Common Lisp -- Language Concepts*. Specifically, the following topics have been improved:

- The topic "**:constructor** Option for **defflavor**" has more detailed information on the parameters of constructors.

- The topic "**:ordered-instance-variables** Option for **defflavor**" now includes an example.

4.2.5.1. Note on make-instance and :fasd-form

Flavor instances are dumped as the forms which are evaluated to create them. As long as the creating forms do not change, flavor instances are compatible across releases and machine architectures. It is always possible that the syntax of **make-instance** will change from one release to another. For this reason, we suggest that you avoid returning **make-instance** from **:fasd-form**, but instead define your own function with a name such as **make-foo-for-loading-from-file** and have **:fasd-form** return a call to that function.

If you later need to change something, you can make your **:fasd-form** start returning calls to a second function such as **make-foo-for-loading-from-file-version-2**. You can keep the previous function **make-foo-for-loading-from-file** around for compatibility with old files, so you can continue to load them.

5. Changes to Zmacs in Genera 7.2

There are two major changes to Zmacs in Genera 7.2.

- You can now create multiple Zmacs frames, each with its own process and its own state, simply by pressing SELECT c-E. With multiple Zmacs processes, SELECT E cycles through the processes.

- There is now a complete Undo facility in the Zwei substrate that can be used from Zmacs, Converse, and the Zmail editors.

In addition there are a number of bug fixes and enhancements.

5.1. Incompatible Changes to Zmacs in Genera 7.2

The old Zwei Undo command and HELP U have been replaced by a full Undo facility. The old commands only remembered one change. The new facility remembers all changes.

The old function **zwei:undo-save** has been removed. Any residual calls to it will provoke a warning at compile time and a recoverable error at run time. In most cases you don't have to do anything to make your changes undoable, as the primitive functions such as **zl-user:insert** and **zl-user:delete-interval** record the changes they make. Obviously if you use **zl:setf** of **aref** to store directly into editor lines, your changes will not recorded. See the section "Zwei Undo Facility", page 27.

The several Find Candidates (m-X) commands have been replaced by one new command, Show Candidates (m-X). Any user-written Zwei commands that call **com-find-any-candidates** will no longer work.

Show Candidates Zmacs Command

Show Candidates (m-X)

Show Candidates prompts for a word or words to search for. By default it performs a heuristic or "smart" search for matching candidates.

With a numeric argument, Show Candidates prompts for a word or word to search for and then asks you to specify the style of matching. Your choices are

- *Heuristic* matching, which is the default. This uses the words you have supplied as *stems*, so that searching for "local" also finds "locative" and "location", for example.

- *Exact* string matching, which means that "local" finds only "local".

- *Initial* exact string matching, which means that "local" finds "local" only in the initial position.

- *Substring* exact string matching, which means that "local" anywhere in the string is matched.

With a numeric argument, you are also asked if you want *adjacent*-word-order matching or *any*-word-order matching if you type more than one word. Any order matching is the default.

The difference between "adjacent" and "any" in the above is that a adjacent-word-order search on "input editor" will find "Using the Input Editor" and "The Input Editor Program Interface" but not "Editor Input" whereas with any-word-order, it will find all three.

This command is also available in the Document Examiner.See the section "Show Candidates Command" in *Genera User's Guide*.See the section "Document Examiner" in *Genera User's Guide*.

5.2. Improvements to Zmacs in Genera 7.2

5.2.1. New Speller Features in Genera 7.2

Spell Word (m-X) now reminds you of m-$ when you type the command without supplying a word to be spelled. Reminder:

- Use Spell Word to check the spelling of a word not in the buffer.

- Use m-$ (Spell This Word) to check the spelling of the word nearest point, or of words in a region.

m-$ now checks the spelling in the region if one is selected, just like Spell Region (m-X). Otherwise, m-$ checks the spelling of the word next to point, as always.

When checking the spelling of a single word, m-$ now informs you if the spelling of the word has been checked against a site-specific or user-specific dictionary. If a word has been checked against the basic Speller dictionary, the following message appears in the minibuffer:

```
"tatterdemalion" is spelled correctly.
```

If a word has been checked against a site-specific or user-specific dictionary, the following message appears in the minibuffer:

```
"Wollongong" is spelled correctly (according to TRADEMARKS).
```

This makes Delete Word from Spell Dictionary (m-X) more useful. For more information about speller dictionaries: See the section "Speller Dictionaries" in *Text Editing and Processing*.

Several thousand words have been added to the basic dictionary, among them tatterdemalion, magnetohydrodynamic, android, hypersphere, yttrium, and burrito. The names of Lisp symbols, such as **defstruct** and **cons**, have also been added.

New documentation giving a recipe for creating speller dictionaries has been added.

5.2.1.1. Quick and Dirty Guide to Adding and Maintaining a Spell Dictionary

Here's how to add your own spell dictionary and keep it up to date:

1. Go into Zmacs and use c-X c-F to create a new file called spell.dict in your top-level directory.

2. Type in any words you know you want to have in your dictionary, or none at all, if you prefer.

3. Write the file out using c-X c-S.

4. If the file is quite long, use Compile Spell Dictionary (m-X) to make a binary dictionary, which loads faster.

5. Use Read Spell Dictionary (m-X) to read the new dictionary in this time. Henceforth, it will be read in when you log in.

6. As you use the Speller, you can add words to your dictionary from the Speller Menu by clicking on the dictionary name, or with Add Word to Spell Dictionary (m-X).

7. To save those new words, use Save Spell Dictionary (m-X).

If you begin to suspect that you have misspelled words in your spell dictionary, here's what to do:

1. Issue the Zmacs command Execute Command Into Buffer (m-X) in an empty Zmacs buffer.

2. Issue Show Contents of Spell Dictionary (m-X). The dictionary is written out into the buffer.

3. Find the misspelled words.

4. Issue Delete Word From Spell Dictionary (m-X) for each misspelled word.

5. Issue Save Spell Dictionary (m-X) to save the corrected dictionary.

This is deliberately simplified information on adding a dictionary and keeping it up to date. For complete information on speller dictionaries: See the section "Speller Dictionaries" in *Text Editing and Processing.*

5.2.2. Fix for Saving a Buffer to a Nonexistent Directory

Formerly, when you wrote a buffer (first time) to a nonexistent directory, you were required to follow a confusing sequence of proceed options, in which you first created the directory, which returned to the debugger. Furthermore, when you saved a buffer, but an underlying directory had been deleted, you got an error message.

In both of these cases, Zmacs now queries, offering to create the directory or abort the command.

5.2.3. Fixes to List Callers

List Callers (m-X) used to not find callers of some functions whose names were lists, because they were recorded as callers of a symbol that was an element of the list, or in the case of **defmacro-in-flavor**, they were not recorded at all. This has been fixed.

Combined methods are no longer recorded as callers because everything they do is determined by wrappers, whoppers, and method combination, and because they do not have source code that you can edit.

5.2.4. Source Compare Can Now Ignore Whitespace or Case and Style

With numeric arguments Source Compare (m-X) can now ignore leading whitespace or case and style in comparing buffers or files.

5.2.4.1. Source Compare

Source Compare (m-X)

Compares two files or buffers, prompting for type (F or B) and name of each, and displays the results of the comparison in the typeout window.

You can modify the behavior of the command with numeric arguments:

- 2 means ignore case and style in making the comparison
- 4 means ignore leading whitespace
- 6 means ignore case, style, and whitespace

Source Compare saves the output in a support buffer named *Source-Compare-*N*. You can read the comparison while checking the file, for example, by going into two window mode with the comparison in one window and the file in the other.

Example

This example shows a comparison between the file bottomley.lisp, as it was read into the buffer, and the buffer horatio.lisp, which contains the contents of the file new *plus* changes that have been made:

```
Source compare made by cheapjack on 12/17/87 14:59:20          -*-Fundamental-*-
of File S:>cheapjack>bottomley.lisp.1
with Buffer horatio.lisp >cheapjack S:
**** File S:>cheapjack>bottomley.lisp.newest, Line #4

  (setq TV:*wholine-clock-delimiters* nil)      ; Kill the "[]" around the clock
**** Buffer horatio.lisp >cheapjack S:, Line #4
  (setq tv:*mouse-exit-target-global-enable* nil)  ; Scroll blobs must die
  (setq TV:*wholine-clock-delimiters* nil)      ; Kill the "[]" around the clock
***************

Done.
```

When entered with a numeric argument of 6, Source Compare ignores both leading whitespace and case and style.

```
Source compare made by robin-hood on 12/17/87 15:45:09          -*-Fundamental-*-
of Buffer bud-abbott
with Buffer lou-costello
No differences encountered.
Source compare made by robin-hood on 12/17/87 15:51:34          -*-Fundamental-*-
of Buffer lou-costello
with Buffer bud-abbott
No differences encountered.
```

When entered with a numeric argument of 4, Source Compare ignores only leading whitespace.

```
Source compare made by robin-hood on 12/17/87 15:54:20          -*-Fundamental-*-
of Buffer bud-abbott
with Buffer lou-costello
**** Buffer bud-abbott, Line #11
  (si:cp-on si:*cp-dispatch-mode* 'si:arrow-prompt)
**** Buffer lou-costello, Line #11
  (SI:CP-ON SI:*CP-DISPATCH-MODE* 'SI:ARROW-PROMPT)
***************

Done.
```

When entered with a numeric argument of 2, Source Compare ignores only case and style.

```
Source compare made by robin-hood on 12/17/87 15:53:38          -*-Fundamental-*-
of Buffer bud-abbott
with Buffer lou-costello
**** Buffer bud-abbott, Line #5
 (setf si:*kbd-auto-repeat-enabled-p* t)

**** Buffer lou-costello, Line #5
             (setf si:*kbd-auto-repeat-enabled-p* t)

***************

Done.
```

5.3. New Features in Zmacs in Genera 7.2

5.3.1. Multiple Zmacs Processes New In Genera 7.2

Genera 7.2 supports multiple Zmacs frames, each with its own process and its own state. You can create new Zmacs processes simply by pressing SELECT c-E. With multiple Zmacs processes, SELECT E cycles through the processes.

5.3.2. New Zwei Undo Facility

There is now a complete Undo facility in the Zwei substrate that can be used from Zmacs, Converse, and the Zmail editors.

The simplest operation of the Undo facility is to undo the most recent change to the editor buffer. Go to a buffer, type something in, delete it, and then press c-sh-U. The deletion is undone. Now press c-sh-R. You're back where you started. Keep pressing c-sh-U. Previous changes to the buffer are undone. You can keep doing this until the buffer is returned to its original state.

5.3.2.1. Zwei Undo Facility

Introduction to Undoing

The Zwei Undo facility remembers all the changes that you have made in an editor buffer and allows you to selectively undo any or all of the changes you have made. The Undo facility is available from Zmacs, Converse, the Zmail draft editor, and other editor buffers based on the Zwei substrate. (It is not available from the Input Editor or in the minibuffer.)

The simplest operation of the Undo facility is to undo the most recent change to the editor buffer. Go to a buffer, type something in, delete it, and then press c-sh-U. The deletion is undone. Region marking shows what was undone. Now press c-sh-R. You're back where you started. It is always safe to undo, because you can always redo, and *vice versa.*

The Undo (m-X) and Redo (m-X) commands are similar to c-sh-U and c-sh-R with the added feature that a display in the minibuffer shows you what will be undone or redone before any action is taken. HELP U also displays the change before undoing it.

Keep pressing c-sh-U. Previous changes to the buffer are undone. You can keep doing this until the buffer is returned to its original state. When you reach this point, if the buffer contains a file, it's no longer marked as needing to be saved. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing to be compiled.

And, repeated pressing of c-sh-R will successively restore the buffer until all the undo commands have been cancelled out.

If you read in a file with no intention of changing it and accidentally type some characters into it, use c-sh-U rather than RUBOUT to get rid of them. That way, the buffer is no longer considered to be modified.

Undo commands operate only on the current buffer. Each buffer has an undo history, and a separate redo history. The undo history can be displayed with c-0 c-sh-U. Likewise, the redo history can be displayed with c-0 c-sh-R. Items in the history are mouse-sensitive. You can undo or redo all changes *up through a given change* or you can undo or redo *any single change* in the history. By default, both histories are discarded when you save the buffer.See the section "Discard Change History", page 34.

Of course, subsequent changes may depend on the single change that you are undoing or redoing, so no guarantee can be made that undoing change number 13 in a 29-change history will have no effect on changes 14 through 29. (On the other hand, you can always back out of any undo or redo. It *is* the Undo facility after all.)

This sounds more complicated in writing than it is when you are doing it. A few minutes experimentation in an editor buffer will make you a competent and confident user of the most important and common undoing and redoing operations.

After an undo or redo, the text that was modified is highlighted the same as if you had marked a region, but in this case there is no region, and the highlighting disappears when you type the next command. The history also shows you what constitutes each change. See the section "What Exactly is a Change to the Undo Facility?", page 28.

What Exactly is a Change to the Undo Facility?

To the Undo facility, a change is a *recorded* change to the textual contents of an individual buffer. Each buffer has its own undo and redo histories of recorded changes.

Most common changes are recorded. You can undo

- Insertions into a buffer, including Execute Into Buffer and Evaluate [and Replace] Into Buffer (m-X).

- Deletions from a buffer, including RUBOUT, m-RUBOUT, c-D, m-D and c-W .
- Text modifications, such as done by m-Q (fill paragraph) or c-TAB (indent differently).

In general, the way changes are recorded is to record the previous contents of the part of the buffer that was changed. Longer changes are compressed to minimize the amount of storage space.

Certain changes are not recorded by the Undo facility. You can't undo

- Renaming a file.
- Compiling a function.
- Reading a file into a buffer.
- Moving around in the buffer.
- Patch system insertions into patch buffers (and other changes to buffers made by the system).
- Revert Buffer (m-X).
- Anything in support buffers such as *Callers-17* or *Definitions-1*.
- Source Compare Merge (m-X) family of commands.

The Undo (m-X) and Redo (m-X) commands and HELP U all display the change before making it. You can also see information about changes in the undo and redo histories displayed by c-0 c-sh-U and c-0 c-sh-R.

You can get a good idea of what a change is by looking at the region marking that you see after an undo. Not all undos result in a clearly marked region — an undo of an insertion does not, for instance — but a line in the minibuffer reports the nature of each undo or redo.

The Undo facility uses simple rules to define changes. Changes are separated by blank lines, paragraphs, if you will. When you type or delete a single character in a new paragraph, that single character is a change. If you then type another character, the two characters together are a change, and so on until you reach the end of the paragraph or expression, that is, another blank line.

Likewise, successive deletions are merged, as are multiple simple commands operating on a single area of text.

There are variables to precisely define all aspects of a change. See the section "Customizing the Undo Facility", page 34.

Undo and Redo Commands

The undo and redo commands come in three styles:

- Undo and Redo (m-X) commands. These commands display the change that will be made before making it.
- Quick Undo and Redo commands. These are bound to c-Sh-U and c-Sh-R, respectively.
- Quick Undo and Redo in Region commands. These are bound to m-Sh-U and m-Sh-R, respectively.

Each of these commands takes numeric arguments that allow selective undoing and redoing and also display the undo and redo histories. Use the undo and redo histories to make wholesale changes.

In addition, you can undo with the the HELP U key.

Undo Zwei Command

Undo (m-X)

Undoes a change to the buffer. The change is displayed first.

The first time you issue the command, it undoes the last change to the buffer. The next time you issue it, it undoes the previous change to the buffer. You can continue this until all changes to the buffer are undone and the buffer is considered unmodified. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing compiling.

For an otherwise equivalent accelerated version of this command:See the section "Quick Undo Command", page 31.

If you undo something you didn't intend to, Redo (m-X) redoes any undo. See the section "Redo Zwei Command", page 32.

With no numeric argument, Undo undoes the most recent change that has not already been undone. With a positive numeric argument, Undo undoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as issuing the command *n* times.

With a negative argument, Undo undoes *only* the *n*th most recent change that has not already been undone. This allows undoing changes out of order, but can sometimes surprise you when the undone change overlaps with a later change that was not undone. Use at your own risk.

With an argument of 0, Undo displays the undo history. Each change is numbered and described in abbreviated form. The undo history is mouse-sensitive. Clicking Left undoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle undoes *only* the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Undo Command

c-Sh-U Quick Undo

Undoes a change to the buffer. The first time you press it, it undoes the last change to the buffer. The next time you press it, it undoes the previous change to the buffer. You can continue this until all changes to the buffer are undone and the buffer is considered unmodified. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing compiling.

For an otherwise equivalent prompting version of this command: See the section "Undo Zwei Command", page 30.

c-Sh-R redoes any undo. See the section "Quick Redo Command", page 32.

With no numeric argument, c-Sh-U undoes the most recent change that has not already been undone. With a positive numeric argument, c-Sh-U undoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, c-Sh-U undoes *only* the *n*th most recent change that has not already been undone. This allows undoing changes out of order, but can sometimes surprise you when the undone change overlaps with a later change that was not undone. Use at your own risk.

With an argument of 0, c-Sh-U displays the undo history. Each change is numbered and described in abbreviated form. The undo history is mouse-sensitive. Clicking Left undoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle undoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Undo in Region

m-Sh-U Quick Undo in Region

m-Sh-U works just like c-Sh-U except that it is limited to changes in the current region rather than in the current buffer. You are warned if an undo extends past the region. See the section "Quick Undo Command", page 31. There is no prompting (m-X) version of this command.

With no numeric argument, m-Sh-U undoes the most recent change that has not already been undone. With a positive numeric argument, m-Sh-U undoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, m-Sh-U undoes *only* the *n*th most recent change that has not already been undone. This allows undoing changes out of order, but can sometimes surprise you when the undone change overlaps with a later change that was not undone. Use at your own risk.

With an argument of 0, m-Sh-U displays the undo history of the region. Each change is numbered and described in abbreviated form. The undo history is mouse-sensitive. Clicking Left undoes *back to* the previous change, just as in using

a positive numeric argument. Clicking Middle undoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the region affected by the change.

Redo Zwei Command

Redo (m-X)

Redoes a change to the buffer. The change is displayed first.

The first time you issue the command, it redoes the last undo in the buffer. The next time you issue it, it redoes the previous undo in the buffer. You can continue this until all changes to the buffer are redone and the buffer is back where you started from before you did the first undo.

For an otherwise equivalent accelerated version of this command: See the section "Quick Redo Command", page 32.

If you redo something you didn't intend to, Undo (m-X) undoes any redo. See the section "Undo Zwei Command", page 30.

With no numeric argument, Redo redoes the most recent change that has not already been redone. With a positive numeric argument, Redo redoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as issuing the command *n* times.

With a negative argument, Redo redoes *only* the *n*th most recent change that has not already been redone. This allows redoing changes out of order, but can sometimes surprise you when the redone change overlaps with a later change that was not redone. Use at your own risk.

With an argument of 0, Redo displays the redo history. Each change is numbered and described in abbreviated form. The redo history is mouse-sensitive. Clicking Left redoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle redoes *only* the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Redo Command

c-Sh-R Quick Redo

Redoes a change to the buffer. The first time you press it, it redoes the last undo in the buffer. The next time you press it, it redoes the previous undo in the buffer. You can continue this until all changes to the buffer are redone and the buffer is back where you started from before you did the first undo.

For an otherwise equivalent prompting version of this command:See the section "Redo Zwei Command", page 32.

c-Sh-U undoes any redo. See the section "Quick Undo Command", page 31.

With no numeric argument, c-Sh-R redoes the most recent change that has not already been redone. With a positive numeric argument, c-Sh-R redoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, c-Sh-R redoes *only* the *n*th most recent change that has not already been redone. This allows redoing changes out of order, but can sometimes surprise you when the redone change overlaps with a later change that was not redone. Use at your own risk.

With an argument of 0, c-Sh-R displays the redo history. Each change is numbered and described in abbreviated form. The redo history is mouse-sensitive. Clicking Left redoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle redoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Redo in Region

m-Sh-R Quick Redo in Region

m-Sh-R works just like c-Sh-R except that it is limited to changes in the current region rather than in the current buffer. You are warned if a redo extends past the region.See the section "Quick Redo Command", page 32. There is no prompting (m-X) version of this command.

With no numeric argument, m-Sh-R redoes the most recent change that has not already been redone. With a positive numeric argument, m-Sh-R redoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, m-Sh-R redoes *only* the *n*th most recent change that has not already been redone. This allows redoing changes out of order, but can sometimes surprise you when the redone change overlaps with a later change that was not redone. Use at your own risk.

With an argument of 0, m-Sh-R displays the redo history of the region. Each change is numbered and described in abbreviated form. The redo history is mouse-sensitive. Clicking Left redoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle redoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the region affected by the change.

The Undo and Redo Histories

The Undo facility keeps an undo history and a redo history for each Zwei buffer. By default, the history is discarded when you save the buffer, but you can set the variable **zwei:*discard-change-record-after-saving*** to **nil** if you wish to override that behavior. There is also a Discard Change History (m-X) command.

You can display the Undo history by giving an argument of 0 to the Undo (m-X) command or the c-Sh-U commands. A 0 argument to m-Sh-U displays the undo history of the current region.

An argument of 0 to the Redo (m-X) or c-Sh-R displays the redo history of the buffer. A 0 argument to m-Sh-R displays the redo history of the current region.

Histories are mouse-sensitive. Clicking Left on a history entry undoes or redoes *back to* the previous change. Clicking Middle undoes or redoes *only* the highlighted change. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Discard Change History

Discard Change History (m-X)

Throws away both the Undo and Redo histories of the current buffer. The buffer is still considered modified after this command is executed, but the histories are gone.

See the variable **zwei:*discard-change-record-after-saving***, page 35.

Customizing the Undo Facility

You can customize the Undo facility by setting one or more variables. Most of these variables affect the way the Undo facility records which changes it can undo or redo, but there are also variables for turning the Undo facility off, for turning off the region marking, and for changing the way the Undo facility handles yanking and multiple replaces.

You can set any of these variables in Lisp programs, including your lispm-init file. You must use **setf**, not **setq** to set these variables.

A number of these variables are also defined as Zwei variables and can be set from inside an editor using the Set Variable (m-X) command. Using the Set Variable command, you can set each variable per buffer, per mode, or globally.

zwei:*enable-change-recording* *Variable*

Set to **nil** to turn off the Undo facility. The default is **t**.

zwei:*undo-sets-region* *Variable*

If **t**, undo and redo operations use region marking to highlight where the change was done. The marking disappears with the next keystroke. If **nil**, there is no marking.

The Zwei variable is called Undo Sets Region. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*yank-is-separately-undoable* *Variable*

If **t**, a yank is separately undoable from any edits before or after it. If **:multi-line**, which is the default, this is true only when the yanked text is more than one line. If **nil**, a yank can be merged with edits before or after it, approximately the same as if the yanked text had been typed in one character at a time.

The Zwei variable is called Yank is Separately Undoable. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*discard-change-record-after-saving* *Variable*

Set to **nil** to cause the Undo facility to *keep* the change histories when a buffer is saved. The **nil** setting conses a lot of garbage and may cause results contrary to user expectations. The default is **t**, meaning that change histories are cleared when a buffer is saved.

The Zwei variable is called Discard Change Record After Saving. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*simple-change-contiguity-range* *Variable*

Default = 3. This sets the maximum numbers of unchanged characters between sections of a simple change *within a line.* If more unchanged characters than this intervene between two changes, they will be considered two undoable changes. Fewer, and it's all one change.

zwei:*simple-change-size* *Variable*

Default = 50. This is the *maximum* number of characters in a change. Once a change is this number of characters, a new change is begun. Changes can be smaller than this number of characters. Edits at the end of such a change are considered part of the same change, but edits in the middle start a new change.

The Zwei variable is called Simple Change Size. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*insertion-amendment-size* *Variable*

Default = 10. This many or fewer characters are considered an amendment to the previous insertion and not a new change, and therefore are cannot be undone separately.

The Zwei variable is called Insertion Amendment Size. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*insertion-breakup-lines* *Variable*

Default = 4. Two insertions of at least this many lines with a blank line between them are considered two undoable changes. In other words, this is the size of an undoable paragraph insertion.

The Zwei variable is called Insertion Breakup Lines. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*undo-each-replace-separately* *Variable*

If **nil**, which is the default, all the changes made by a Replace String (c-?) or Query Replace (m-?) or other member of the Query Replace (m-X) family are undone as a unit. If **t**, the changes can be undone separately. Setting this variable is a matter of taste and style.

The Zwei variable is called Undo Each Replace Separately. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

5.4.1.8. Going Back to First Indented Character

m-M
c-m-M
m-RETURN
c-m-RETURN
Back To Indentation

Positions point before the first nonblank character on the current line.

5.4.1.9. Deleting Indentation

m-^
c-m-^
Delete Indentation

Deletes the newline character and any indentation at the beginning of the current line. It tacks the current line onto the end of the previous line, leaving one space between them when appropriate, for example, at the beginning of a sentence.

With any numeric argument, it moves down a line first, killing the whitespace at the end of the current line.

5.4.1.10. New Line with This Indentation

m-O
This Indentation

Makes a new line after the current one, deducing the new line's indentation from point's position on the current line. If point is to the left of the first nonblank character on the current line, it indents the new line exactly like the current one. But if point is to the right of the first nonblank character, it indents the new line to the current position of point. Regardless, it leaves point at the end of the newly created line.

With a numeric argument, the new line is always indented like the current one, no matter where point is. With an argument of zero, it indents current line to point.

5.4.1.11. Moving Rest of Line Down

c-m-O
Split Line

Moves rest of current line down one line. It inserts a carriage return and indents new line directly beneath point. With a numeric argument *n*, it moves down *n* lines.

5.4.1.12. Inserting Blank Line

c-O
Make Room

Inserts a blank line after point. With a numeric argument *n*, it inserts *n* blank lines.

5.4.1.13. Deleting Blank Line

c-X c-O
Delete Blank Lines

Deletes any blank lines around the end of the current line.

5.4.1.14. Centering the Current Line

m-S Center Line

Centers the text of the current line within the line. With an argument *n*, it centers *n* lines and moves past them. Do not use this command for indenting Lisp code.

5.4.1.15. Controlling Indentation of Lisp Forms

This section shows you how to control the way that Zmacs indents Lisp forms when you use LINE, TAB, c-m-\, c-m-Q and so forth. This information is most useful when you are creating your own macros, which parse their arguments differently from functions.

This information applies to Zmacs in Lisp mode only.

In indenting forms, Zmacs makes no distinction beween functions, macros, or just data. Indentation is controlled by a property of the first symbol of the form. For convenience, this discussion refers to these symbols as functions if the technique applies to symbols of any type and as macros when the technique applies to macros only.

There are four methods of controlling indentation:

1. Start the name of your function with **def...**
2. Use **&body** in your macro's argument list
3. Use the **zwei:indentation** declaration
4. Use the **zwei:defindentation** special-form

All four methods can be used to control the indentation of macros. The **def...** method and the **zwei:defindentation** method can be used to control the indentation of other forms as well. If you use none of these methods, forms beginning with your symbol will indent like ordinary functions.

It should be noted that Zmacs will not know that you have used **zwei:indentation** or **&body** until you compile your **defmacro** or evaluate the **zwei:defindentation**.

This discussion of indentation uses some special terminology.

For instance, *indentation n* means that Zmacs indents the first character of that argument *n* characters to the right of the first character of the function name. Thus, if Zmacs indents a form like this

```
.... (elbo-macro
      charm
       "grace"
        (beauty))
```

then the argument charm is getting indentation 0, the argument "grace" is getting indentation 1, and the argument (beauty) is getting indentation 2.

Standard indentation refers to the indentation behavior of elements of normal lists. The first form on a line indents to the same column as the first argument on the previous line which belongs to the same function. When the first argument of a function is not on the same line as the function name, it gets indentation 1. Here are some examples of standard indentation:

```
(list 0
      1
      2)

(list
  0
  1
  2)

(list 0 1
      2 3 4)
```

Controlling Indentation by Naming Your Function def...

If the name of your function begins with **def...**, then argument 2 (the "third" argument) is given indentation 1, and all other arguments get standard indentation, like **defun**.

For example:

```
(defmacro define-thing (name args thing-maker thing-destroyer)
  `(progn (defun (:property ,name maker) ,args ,thing-maker)
          (defun (:property ,name destroyer) ,args ,thing-destroyer)))

(define-thing sylon
              (x y z)
  (...)
  (...))
```

This indentation behavior is overridden by use of **&body**, **zwei:indentation**, or **zwei:defindentation**.

Controlling Indentation Using &body

When you use **&body** in the argument list of a macro, the first **&body** argument gets an indentation of 1. All other arguments get standard indentation.

For example:

```
(defmacro macro-with-body-arg (a b &body body)
  `(list ,a ,b ,@body))

(macro-with-body-arg 0
                     1
  2
  3)
```

This indentation behavior is overridden by use of **zwei:indentation** or **zwei:defindentation.**

&rest and **&optional** have no effect on indentation.

Controlling Indentation Using zwei:indentation

zwei:indentation *Declaration*

zwei:indentation *subform-index indentation subform-index indentation...* declaration

zwei:indentation . *indentor-function* declaration

The **zwei:indentation** declaration (and the **zwei:defindentation** special form) give you the most control over the way Zmacs indents calls to your macros.

zwei:indentation is placed in the declaration part of the **defmacro** which defines your macro. When using the first syntax of **zwei:indentation**, the declaration should be given an even number of arguments. Each pair of arguments assigns a specific indentation to a particular argument. Note that, for the purposes of this declaration, *the arguments of your macro are numbered from 0.*

Here is an example using **zwei:indentation**:

```
(defmacro strangely-indented-macro (arg0 arg1 arg2 arg3 &rest rest)
  (declare (zwei:indentation 2 3 5 0 6 2))
  `(list ,arg0 ,arg1 ,arg2 ,arg3 ,@rest))
```

This causes argument 2 to get indentation 3, and argument 5 to get indentation 0. All other arguments get standard indentation, which causes argument 1 to follow argument 0, argument 3 to follow argument 2, etc.

Here is how this example macro is indented:

```
;0123456  <- indentation
(strangely-indented-macro 0
                          1
    2
    3
    4
 5
   6
   7
   8
   9)
```

zwei:indentation has a second syntax. When the **cdr** of the declaration is a function or a symbol with a function definition, that function is called to do the indenting.

For example,

```
(declare (zwei:indentation . zwei:indent-prog))
```

would cause the macro to indent like **prog**.

In addition to **zwei:indent-prog**, you could also use **zwei:indent-prog-or-tagbody, zwei:indent-tagbody,** or **zwei:indent-loop.** There is no further documentation on these function definitions.

Controlling Indentation Using zwei:defindentation

zwei:defindentation *Special Form*

zwei:defindentation (*name subform-index indentation subform-index indentation...*)

zwei:defindentation (*name . indentor-function*)

zwei:defindentation is similar to **zwei:indentation** in that it allows you to control the indentation of certain forms. The *subform-index*, *indentation* and *indentor-function* arguments work in the same way as for **zwei:indentation.**

zwei:defindentation differs from **zwei:indentation** in two ways:

- **zwei:defindentation** can be used to control the indentation of forms beginning with any symbol, not just macros. The *name* argument specifies the symbol whose indentation you wish to control.

- **zwei:defindentation** is a special form which must be compiled or evaluated after the definition of your macro, function, or other symbol.

When defining a macro, you will probably find **zwei:indentation** more convenient than **zwei:defindentation** , because there will be one less definition in your source file, and because recompiling your macro will undo any previous indentation specifications for that macro.

This example shows how to use **zwei:defindentation** to control the indentation of a function:

```
(defun funny-function (&rest rest)
  (length rest))

(zwei:defindentation (funny-function 3 5))

(funny-function 0
                1
                2
      3
      4)
```

6. Changes to Utilities in Genera 7.2

6.1. New Features in Utilities in Genera 7.2

6.1.1. New Metering Interface in Genera 7.2

Genera 7.2 offers a set of flexible tools for metering programs.

Metering is the process of measuring the performance of a program, usually with the goal of determining where performance can be improved. Genera offers tools for metering different aspects of performance, such as time, paging, and consing. The Metering Interface is a uniform interface which makes it convenient to use the various metering tools.

This chapter also documents several macros that are useful for metering short forms. See the section "Macros for Metering the Execution Time of Forms", page 81.

6.1.1.1. Overview of the Metering Interface

This section gives an overview of how to use the Metering Interface. The Metering system is not part of the default world; it is loaded separately. To begin, load the system:

```
Load System Metering
```

Now you can select the Metering Interface:

```
SELECT %
```

The name % was chosen because it is related to metering; you might be seeking the percentage of time spent in one function, or some other percentage. (You can always use `SELECT HELP` to remind yourself of the activities you can choose with the `SELECT` key.)

You will notice that when you first select the Metering Interface, it takes a little while for the Metering Interface to do some preparatory work. The progress note says "Computing Fudge Factors". We suggest that you wait for this to finish before typing, using the mouse, or doing any other activity that would interfere with this computation. See the section "Computing Fudge Factors", page 53.

Metering involves a sequence of steps. Here we briefly describe each step, and refer to the section that describes the step in further detail.

1. Specifying what to meter

   You can meter the performance of a Lisp form, a portion of a function, or one or more functions running within a process. For example, to execute and

meter a form, click on [Meter Form] and enter the form. You could also meter one or more functions running within a process by clicking on [Meter in Process]. See the section "Specifying What to Meter", page 47.

2. Choosing the type of metering

   The Metering Interface prompts you for a metering type. The metering type controls how the data is collected and presented. The choices are: Function Call, Page Fault, Call Tree, Statistical Function Call, Statistical Call Tree, and Statistical Program Counter. See the section "Choosing a Metering Type", page 48.

3. Specifying metering parameters

   This is an optional step. The keyword options to Meter Form and Meter in Process allow you to control various aspects of the metering run. For example, you might decide to meter the code within a **without-interrupts** form, or to run the code a number of times and meter only the results of the last time it executes. When doing a Page Fault metering run, you can flush all pages first. The keywords available depend on which metering command you give, and the type of metering. See the section "Meter Form Command", page 74.See the section "Meter in Process Command", page 76.

4. Running and metering the code

   When using Meter Form, once you choose the type of metering and press RETURN, the form is immediately executed and simultaneously metered.

   When using Meter in Process you can meter one or more functions whenever they are naturally executed within a given process, instead of executing them immediately. This is useful for metering a function that normally runs within a process such as Zmail or a network process; the metering results are more representative of the usual environment of the function than they would be if you called the function explicitly.

   When you use Meter Form or Meter in Process, the result is called a *metering run*, which contains the data collected. The Metering Interface saves a history of metering runs in the top pane, which makes it convenient to show the data of a metering run later, or to repeat the metering run. The "current" metering run is the run whose results are now being displayed. The current metering run appears in bold face in the Metering History. See the section "Running and Metering the Code", page 50.

5. Customizing the display of metering results

   This is an optional step. The display in the bottom pane shows the results of the metering run. The results appear in columns under headers that describe the data. Each column is an *output field.* An output field shows a kind of

data, such as consing, page faults, or time spent in a function. Output fields are divided into subfields; each subfield shows one aspect of the information.

You can click Middle on a column header for information describing the data in that output field. You can also tailor this display to request more or less information by removing output fields from the display or adding them to the display. See the section "Customizing the Display of Metering Results", page 51.

Often the data displayed is only a summary of the data available. You can get expanded information on a particular portion of the data by clicking Middle on it. See the section "Expanding Metering Data", page 61.

6. Interpreting the results of the metering run

   In this step you analyze the metering results and try to identify where the performance of your program can be improved. See the section "Interpreting the Results of a Metering Run", page 55.

7. Saving the results of a metering run

   The results of a metering run are saved in your Lisp world until you explicitly delete the metering run or cold boot. However, sometimes it is useful to save the results more permanently, either by printing them or by sending the results to an editor buffer and then using a Zmacs command to write the results to a file. To do this, use Show Metering Run and give the :Output Destination option. See the section "Show Metering Run Command", page 80.

Usually when you are metering a program, you go through the cycle of metering steps several times. You might choose other metering types to collect information on different aspects of performance. You might modify the program on the basis of the metering results, and then meter the program again.

When you begin using the Metering Interface, you might make use of the Metering Help facilities: See the section "Getting Help in the Metering Interface", page 47. The Metering Interface enables you to use the mouse to give many of the metering commands: See the section "Using the Mouse in the Metering Interface", page 54.

When you do a metering run of a new metering type, it takes some time for the Metering Interface to do the necessary compilation and set-up work. However, future metering runs of the same metering type will not have this start-up delay.

6.1.1.2. How to Use the Metering Interface

Getting Help in the Metering Interface

Here are some suggestions for getting help within the Metering Interface. Most of these suggestions are applicable in most other contexts of the Symbolics Genera environment.

- What metering commands are available? Press the HELP key for a list.
- What does a metering command do? Enter Help *command-name* to see the documentation for a given metering command. You can also click Middle on any of the commands visible in the command menu.
- What operations can you do on a metering run? Click Right on a metering run displayed in the History of Metering Runs pane for a menu of operations.
- What operations can you do on an output field? Click Right on an output field displayed above the Metering Results pane for a menu of operations.
- What operations can you do on a node of a call tree display? Click Right on a node displayed in the Metering Results pane for a menu of operations.
- What is the information presented under a output field or subfield? Click Middle on an output field or subfield to describe its contents.

Specifying What to Meter

The first decision is whether you want to meter within the scope of a form or within a process.

When you use Meter Form, the form is executed in the Metering Interface process, and simultaneously metered. Metering will occur only within that form. You can meter everything occurring within that form, or specify functions or portions of functions to meter within that form. See the section "Meter Form Command", page 74.

Sometimes you want to meter a function whenever it is normally called within a process. You don't want to use Meter Form, because that would execute and meter the function immediately. Meter in Process allows you to meter one or more functions within a process, without explicitly calling those functions. You can meter everything occurring within the process, or specify functions or portions of functions to meter within the process. See the section "Meter in Process Command", page 76.

For both Meter Form and Meter in Process, you will be prompted for "What to meter", which allows you to further specify the code to be metered. The choices are:

Everything	Meter everything within the form/process.
Only when Enabled	Meter only the code which is surrounded by a **mi:with-metering-enabled** form.

Within Functions — Meter only within the functions specified. You will be prompted for :Metered functions, and you should enter the functions of interest.

Only when Enabled is used to meter only a portion of code. First you edit one or more functions of interest to wrap **mi:with-metering-enabled** around the desired portion or portions of code and compile the changed function (or functions). Then you can use Meter Form or Meter in Process and specify :Only When Enabled. This starts a metering run that will meter only the code in the dynamic scope of **mi:with-metering-enabled**. See the section "Controlling Metering Within Lisp Code", page 69.

Choosing a Metering Type

This section describes each metering type, and then gives some general guidelines and suggestions about choosing the right metering type for different purposes.

Function Call — Collects data on every function entry and exit. The display indexes the data by function. This display shows the number of times each function was called, the total amount of time spent in each function, the total amount of consing that took place in each function, and information on any page faults that occurred in each function.

Call Tree — Collects the same data as Function Call, but the display indexes the data by the stack trace. This describes the entire calling sequence of functions that occurred. Each function is a node of the tree; the callees of a function are displayed below the function and indented. You can selectively conceal or display nodes of the tree.

Page Fault — Collects data related to the paging system. The display indexes the data by page fault. The display shows how much time was spent in each page fault, what kind of page fault occurred, the virtual address and physical page where the page fault occurred, and the function and/or Lisp object whose reference caused the page fault.

Statistical Function Call
: Collects and displays the same kind of data as does Function Call. The difference is that Statistical Function Call does not collect data on every function entry and exit; instead, it periodically samples the process being metered.

Statistical Call Tree
: Collects and displays the same kind of data as does Call Tree. The difference is that Statistical Call Tree does not collect data on every function entry and exit; instead, it periodically samples the process being metered.

Statistical Program Counter

This metering type incorporates the PC Metering Tools into the Metering Interface. It collects and displays only exclusive time, and indexes it by function. It automatically executes the form a number of times, gradually zooming in on the functions where most of the time is spent.

Where to Start?

Function Call and Call Tree are the typical places to start metering. These metering types collect the same kind of data, but they display it differently. Since Function Call indexes the data by function, you can see the total amount of time spent in each function, the total number of times a function was called, and totals for paging and consing during each function. However, the Function Call display does not inform you of the calling sequence. In contrast, Call Tree indexes data by the stack trace, which informs you of the calling sequence. However, if a function was called in more than one place in the call tree, the information on that function is not merged together to show you the total number of times the function was called, total paging, total consing within that function, and so on.

Metering Long Runs

You will notice immediately that when you execute and meter code, it takes much longer than running the code normally (without metering). For a long run, it might be prohibitively time-consuming to meter every function entry and exit.

You can use the Statistical Function Call or Statistical Call Tree metering types to periodically sample the process being metered instead of collected data on every function entry and exit; this enables you to identify performance problems in runs that are too long to meter completely.

Often it is useful to use one of the statistical metering types to get a rough idea of where the performance problems are, and then narrow the scope of the metering to focus on those problems. You can edit your program to use **mi:with-metering-enabled** to specify exactly what code should be metered. You can then use Function Call or Call Tree metering types (and supply the :Only When Enabled keyword as Yes); much less data will be collected, and the metering run will go faster. You can also use **mi:with-metering-enabled** with the statistical metering types.

Function Call metering is significantly faster than Call Tree metering. Function Call and Call Tree metering take much longer than their statistical counterparts, but the data is deterministic, whereas the result of statistical metering is statistical data.

Metering for Paging Performance

The Function Call and Call Tree metering types give information on paging, including the number of page faults and the time spent in the paging system. Often that information is exactly what you are looking for, and there is no need to use Page Fault type of metering.

Page Fault metering collects and displays more detailed information on page faults, and the activities of the paging system. It is intended for people already familiar with paging systems. Page Fault metering shows what function or object took the page fault. It also shows information about fetching that occurred, which is useful for programmers who control the prefetch count by using the **:swap-recommendations** option to **make-area**.

Integration of PC Metering into the Metering Interface

The Statistical Program Counter metering type integrates the PC Metering tools into the Metering Interface. See the section "PC Metering" in *Program Development Utilities*.

PC Metering was available prior to the Metering Interface, which was introduced in Genera 7.2. Probably for most purposes the other types of metering will collect and display the desired kinds of data.

The Statistical Program Counter metering is useful only when the form you are metering has strictly repeatable results. (You cannot use this type of metering for Meter in Process, only for Meter Form metering runs.) The form is executed a number of times. It collects and displays the percentage of exclusive time spent in functions; this information is indexed by function.

In Statistical Program Counter metering, the sampling is supported by microcode. This means it can meter code within a **without-interrupts** special form. In contrast, in Statistical Function Call and Statistical Call Tree metering, the sampling is done from another process, so it cannot take place within **without-interrupts**. Also, the data will show time spent in escape functions, which is not shown in the other metering types.

Running and Metering the Code

When using Meter Form, once you have entered the arguments and pressed RETURN, the metering run begins. The form is executed and the desired kinds of data are collected during the execution. A metering run takes significantly longer than running the code without metering, because metering collects a lot of data. When the metering run finishes, the results are displayed in the Metering Results pane.

When using Meter in Process, the function being metered is not executed by the Metering Interface. Instead, the Metering Interface meters that function within some other process. For example, you might want to meter a function that normally runs within a network process; you would not want to call that function explicitly, but rather meter it whenever it normally is called. Once metering is started, whenever the function is called within that process, it is metered. In Meter in Process you specify explicitly when the metering should start and stop. Whenever you stop the metering, the desired kinds of data are collected into a metering run, which is displayed in the Metering Results pane.

Customizing the Display of Metering Results

The default display of metering results is a summary of the data collected. Each column is an *output field.* An output field shows a kind of data, such as consing, page faults, or time spent in a function. Output fields are divided into subfields; each subfield shows one aspect of the information.

You can request more detail by adding output fields or subfields to the display, or by expanding some piece of data already shown. You can request less detail by deleting output fields or subfields from the display. If desired, you can also set the default output fields for a given type of run, and cause other runs of that type to be displayed using the new defaults.

You can get information on the display itself, such as finding out what units are being displayed. You can use the Metering History to show the results of a previous metering run.

In many cases, you can give the following commands by using a mouse gesture. See the section "Using the Mouse in the Metering Interface", page 54.

Getting Information on the Display Itself

You can describe the meaning of a major output field or a subfield by positioning the mouse over the field and clicking Middle.

Describe Output Field Command
: Describes the meaning of the data displayed in a given output field. You can do this by clicking Middle on an output field.

Displaying Information not Currently Visible

You can request expanded data by clicking Middle on a piece of data. You can scroll the various window panes by positioning the mouse on the scroll bar and using the normal scrolling commands.

Expand Field Command
: Expands the data identified by the output field (column) and the function (row), for a given metering run. You can do this by clicking Middle on the piece of data you want to expand.

Deleting Output Fields from the Display

You can delete a major field or a subfield from the display by positioning the mouse on the field and clicking sh-(M).

Delete Output Field Command
: Deletes an output field from the display of the metering run. You can do this by clicking sh-Middle on the output field you want to delete.

Delete Output Subfield Command
: Deletes an output subfield from the display of the metering run. You can do this by clicking sh-Middle on the output subfield you want to delete.

Adding Output Fields to the Display

Add Output Field Command
: Adds a new field to the display of the metering run. You can do this by clicking c-m-Left on a metering run.

Add Output Subfield Command
: Adds a new subfield to the display of a metering run. You can do this by clicking c-m-Left on an output subfield.

Freezing the Display while Adding or Deleting Fields

Lock Results Display Command
: Prevents updating of the display of metering results until the Unlock Results Display command is given. Useful when you are adding or deleting several output fields.

Unlock Results Display Command
: Re-enables the updating of the display of metering results. Use this after you have used Lock Results Display and finished customizing the output fields.

Changing the Defaults for Displaying

Once you have added or deleted fields from a metering run, you might want to cause all future metering runs of that metering type to display the same fields that this run displays. To do so, use Set Default Output Fields for Type. That sets the default output fields for displaying runs of that metering type, but it only affects future metering runs. You can use Set Output Fields of Run from Defaults to cause an existing metering run to use the new defaults.

Set Default Output Fields for Type Command
: Sets the defaults for displaying future metering runs to be the same as the fields displayed for the given metering run. This only affects the display of metering runs of the same metering type as this run.

Set Output Fields of Run From Defaults
: Sets the output fields of the given metering-run to the defaults. This is useful when you have changed the defaults and you want a metering run to use the new defaults.

Using the Metering History

You can position the mouse over a metering run in the Metering History. Then you can click Left to display the results of the run, or Middle to describe the run, or Right for other alternatives.

Describe Metering Run Command
: Describes a metering run, including the date and time of the run, what code was metered, and the metering parameters that were used. You can do this by clicking Middle on a metering run.

Show Metering Run Command
: Displays the results of a metering run. You can do this by clicking Left on a metering run.

Delete Metering Run Command
: Deletes a metering run from the Metering History. You can do this by clicking sh-Middle on a metering run.

Re-Meter Command
: Repeats a metering run, selecting the type of metering and the code to meter from the specified metering run. You can do this by clicking s-Middle on a metering run.

Changing Other Aspects of the Display

Move Output Field Command
: Moves an output field to another position in the display of metering results. You can do this by clicking c-m-Middle on an output field.

Set Display Options Command
: Enables you to specify how the data of a metering run should be displayed, including how the data should be sorted. You can do this by clicking on [Set Display Options] in the menu.

Set Indentation Depth Command
: Specifies how many levels not to indent for displaying a call tree metering run. You can do this by clicking s-m-Middle on a displayed node, to start indentation after that node.

For information on customizing the display of a call tree metering run: See the section "Exploring a Call Tree", page 56.

Computing Fudge Factors

When you first select the Metering Interface, some initialization work goes on. The progress note says "Computing fudge factors." Here "fudge factors" are based on the hardware and software configuration of your machine. The goal of this computation is to measure the overhead of the metering tools.

We recommend that you wait until this process has completed before you type anything. It is important for the machine to be otherwise idle, while the fudge factors are being calculated.

The computation happens more than once, and if the results are quite similar, the Metering Interface considers the fudge factors to be consistent and reliable. If the results vary significantly, the computation is believed to have failed. This can happen if you move the mouse rapidly during the computation, for example, or if something else requires action on the part of the machine, such as unusually heavy network traffic.

When the fudge factors have not been calculated accurately, any metering results you later obtain will not be accurate. Incorrect fudge factors can result in negative times for short functions, for example.

When the fudge factor computation fails, the Metering Interface prompts you for what to do. The choices are:

Retry once	Make one more attempt to do the computation, and prompt again if it fails.
Retry	Continue retrying the computation until the measurements are consistent.
Ignore	Use the values computed so far, even though they are possibly inconsistent.

We advice retrying the computation. To make it more likely to succeed, you might try moving your mouse off the screen, make sure the garbage collector is off, or wait for network traffic to die down.

If this is not possible, you can use Ignore to proceed past this stage. However, we recommend against using Function Call or Call Tree metering unless the fudge factors have been computed correctly.

Before using metering types Function Call or Call Tree you can recompute the fudge factors by evaluating the following form:

```
(progn
 (setq metering:*function-entry-fudge-factor-1* 0)
 (metering:enable-metering-utility))
```

Using the Mouse in the Metering Interface

It is often convenient to use the mouse to give commands in the Metering Interface. This section summarizes the available mouse gestures. The mouse gestures are arranged in patterns so it should be easy to remember how to use them. For example, clicking Middle on something describes that thing.

To take action on a metering run, you can click on a metering run in the Metering History. To indicate the current metering run, you can click on the header of the Metering Results pane (where the names of the output fields appear).

Mouse Gesture	*Action*
Left	Says "do it". When used on a metering run, it displays the results of the run. When used on a metering command in the menu, it prompts you for the arguments to the command. This mouse gesture can mean different things in different contexts; it usually enables you to do the most commonly done action on the highlighted thing. When used on a node in a call tree display, it offers to hide or show the children (whichever is appropriate).
Middle	Gives a description. Can be used on a metering run, a metering command in the menu, an output field or subfield, or a piece of data (to expand the data).

Right	Gives a menu of commands that can be given on the highlighted thing. Can be used on a metering run, an output field or subfield, or a visible node in a call tree display.
sh-Middle	Deletes the highlighted thing. Can be used to delete a metering run, to delete an output field or subfield, or to hide a node in a call tree display.
c-m-Left	Adds an output field or subfield. When used on a metering run, it adds an output field. When used on an output field, it adds an output subfield.
c-m-Middle	Moves an output field or subfield.
s-Middle	On a metering run, re-meters the run.
s-Left	On a node in a call tree display, shows all the node descendants.
s-m-Left	On a node in a call tree display, hoists the node. On a node that has already been hoisted, dehoists the node.
s-m-Middle	On a node in a call tree display, starts the indentation after that node.

The Metering Interface also offers the following command keyboard accelerators, which are based on similar accelerators in other parts of Genera:

c-sh-D	Describes the current metering run. (This is based on c-sh-D, which means describe in Zmacs and the input editor.)
c-m-R	Re-Meters the current metering run. (This is based on c-m-R, which means re-invoke in the Debugger.)
c-m-U	Dehoist current node. This takes a numeric arg. An integer greater than 0 tells how many levels to dehoist. A numeric argument of 0 Dehoists all the way. The default is 1 level. (This is based on c-m-U in Zmacs and the input editor, which means "up one level of list structure".)

6.1.1.3. Interpreting the Results of a Metering Run

In general, interpreting metering results is a skill that requires practice and familiarity with the code being metered. We suggest that you do metering with specific, limited questions in mind, rather than metering with too great a scope and being overwhelmed with data, much of which is not relevant. Another approach is to start with a general question in mind, and use the metering results to help you limit the scope of future metering runs, thus enabling you to focus on the important aspects of your program's performance.

Be aware that the default display of metering results shows many fields that might be of interest, but for any given metering run, some of those fields may not be of interest. You might find it useful to delete fields from the display in order to focus on the fields that are relevant to the question being asked. On the other hand, you can also add other fields to the display.

This chapter describes the important concepts that apply to interpreting metering results. We do not document here what each of the fields of data means. We suggest that you use the online documentation available within the context of the Metering Interface. To find out what a field of data means, position the mouse over a field or subfield and click Middle to describe it.

Inclusive and Exclusive Time in Metering

When interpreting metering results, it is important to understand the meaning of *inclusive time* and *exclusive time.*

Inclusive time	The amount of time spent in function, *including* time spent in any functions that this function called.
Exclusive time	The amount of time spent in function, *excluding* time spent in any functions that this function called.

The terms "inclusive" and "exclusive" are also applied to other aspects of performance, such as consing or page faults. "Inclusive" always means that any callees of the function are included in the data, whereas "exclusive" means that the callees are excluded from the data.

For an illustration of inclusive versus exclusive time, suppose you meter the form **(format t "~&hello, world.")** and specify the Function Call type of metering. The first function in the display is **format.** The inclusive time of format is very large; in fact it is equal to the amount of time spent in the run. However, the exclusive time of **format** is very small, because most of the time spent in **format** was actually spent in functions called by **format.**

The inclusive time of a function is the sum of the inclusive times of its callees and the exclusive time in itself.

Exploring a Call Tree

For Call Tree and Statistical Call Tree types of metering, the output field labeled Function contains the "call tree" of the functions. This describes the entire calling sequence of functions that occurred. The callees of a function are displayed below the function and indented.

Each function in a call tree display is called a *node.* The first function is called the *root node*; this is the top-level function you metered. The callees of a function are known as the "children" of that node. The descendants of a node include all of its children, all of their children, and so on.

Usually the call tree is not presented in entirety because that would be too long and hard to decipher; instead a heuristic determines which nodes should be displayed. The Metering Interface offers several ways to explore a call tree display. You can open a node (show all of its children) or close it (hide all of its children).

The symbols in the call tree have the following meanings:

↓ This node is completely opened; all of its children are shown.

→ This node is not opened at all; none of its children are shown.

↔ This node is partially open; some of its children are shown.

• This is a terminal or "leaf" node; it has no children.

In many cases, you can give the following commands by using a mouse gesture. See the section "Using the Mouse in the Metering Interface", page 54.

Commands for exposing nodes further

Show Node Children Command
: Adds all the children of a node to a call tree metering display. You can do this by clicking Left on a node with undisplayed children.

Show All Node Descendants Command
: Adds all the descendants of a node to a call tree metering display. You can do this by clicking s-Left on a node.

Commands for concealing nodes or portions of nodes

Hide All But Path to This Node Command
: Customizes a call tree metering display to show only the path to the given node, by removing functions from the display that do not lead directly to this node.

Hide Node Children Command
: Removes all the children of a node from a call tree metering display. You can do this by clicking sh-Left on a node which is partially or completely open.

Hide Node Command
: Removes a node and all of its descendants from a call tree metering display. You can do this by clicking sh-Middle on a node.

Commands for changing the root node

Hoist Node Command
: Changes a call tree metering display to focus on a certain node as if it were the root node, and removes all functions from the display which are not descendants of this node. You can do this by clicking s-m-Left on a node.

Dehoist Command
: After you have hoisted a node, you can use Dehoist to restore the display to a different root node that is no longer displayed. You can do this by clicking s-m-Left on a node that has been hoisted.

When you hoist a node, it is often useful to add the /Root subfield to one or more fields of interest. For example, the /Root subfield of Exclusive Time output field shows the fraction of exclusive time spent in a given function, with respect to the new root (as opposed to /Run, the fraction of time in a given function with respect to the whole run).

Command for altering the indentation

Sometimes the indentation is so great that the names of the functions are pushed off the right edge of the display. There are two solutions to this problem. First, you can scroll the window horizontally by using the usual scrolling commands. Second, you can use the Set Indentation Depth command to specify how many levels of the tree should be displayed without indentation; the following levels will be indented.

Set Indentation Depth Command
: Specifies how many levels not to indent for displaying a call tree metering run. You can do this by clicking s-m-Middle on a displayed node, to start indentation after that node.

Different Views of the Same Metering Data

The Metering Interface can often give you different views of the same data. For example, Inclusive Time is a field that can express its data in several views; each view is expressed by a subfield of the Inclusive Time field. Some of the views include:

Total	total time spent inclusively in the function
/Run	a bar graph showing the proportion of time spent inclusively in this function with respect to time spent in the whole run
%Run	same as /Run, but expressed as a numerical percentage
Avg	average amount of time spent inclusively in this function, per call

Notice that some of the views describe the relationship between two kinds of data. For example, /Run shows the proportion of time spent inclusively in this function with respect to time spent in the whole run.

The default display shows some of these subfields. You can choose to add subfields to the display, or delete subfields from the display. See the section "Customizing the Display of Metering Results", page 51.

Often the metering results displays a summary of the collected data, and additional data is available to you. You can position the mouse over a piece of data, and click Middle to expand it. See the section "Expanding Metering Data", page 61.

Overview of How Metering Works

This section briefly summarizes how metering works, which should help you understand what the results mean. The metering substrate is the implementation underlying the Metering Interface.

Background on Function Call and Call Tree Metering

When you start metering something, the metering substrate sets up a trap which is entered when the code to be metered begins to execute. When this trap is entered, the metering substrate notes the time when the the function begins to run; it also begins to collect data on paging and consing. When the code being metered finishes, the trap is exited and the metering substrate notes the time when it ended (and other data).

You will notice that it takes longer to execute and meter a form than it does to execute the form without metering it. However, it is important to note that the metering substrate subtracts all of its own overhead from the metering results. That is, the metering results (time, page faults, paging system, and consing) correctly exclude any work done by the metering substrate itself.

Results Collected on a Per-process Basis

Note that during the metering, the scheduler might switch processes from the process in which the metered code is running to some other process. When this happens, the metering substrate "turns off" the collection of data on page faults, paging system, and consing. Thus the page faults, paging system, and consing results are collected on a per-process basis, and they correctly exclude any page faults, paging, or consing done within a different process. However, this is not the case for the data on time: See the section "How Time Spent in Other Processes Affects Metering Results", page 64.

Metering Overhead is Excluded when Possible

One overall design goal of the metering tools was to subtract out overhead only if it could be done accurately, and when this is not possible, to document the possible anomalies in the metering results. The alternative would be to attempt to estimate the overhead, which might yield incorrect results, without the user being aware that the results were inaccurate. One example of the metering tools not subtracting out overhead is in sequence breaks.
See the section "The Effects of Sequence Breaks on Metering Results", page 66.
See the section "Metering Percentages Greater Than 100%", page 60.
See the section "Metering Overhead When :Within Functions is Used", page 60.

Background on Statistical Program Counter Metering

The Statistical Program Counter metering type (also called PC Metering) is done at the microcode level, and it works differently than the other metering types. For details: See the section "PC Metering" in *Program Development Utilities*.

PC metering divides up compiled functions into "buckets" by their locations in memory. It repeatedly executes the form provided to Meter Form, sampling the PC at a high rate. It increments the count of a bucket each time a PC falls within that range. If the number of samples in a bucket is greater than a given percentage (the *resolution percentage*) of the total number of samples, it will rerun your form and "zoom in" on this particular bucket. It "zooms in" by ignoring all PC's

outside of the bucket of interest, and therefore is able to use progressively finer and finer resolution buckets. When a bucket contains a single function the "zooming" stops.

The resolution percentage controls how many buckets the metering interface "searches" (it will skip all buckets that take up less than the resolution percentage of the total), and consequently how many times it must repeat your form. The finer (or smaller) the resolution, the more times it will have to repeat your form in order to investigate more buckets.

Metering Percentages Greater Than 100%

For the output fields that give information in a ratio or percentage, such as the /Run subfield of Inclusive or Exclusive Time, sometimes the result is greater than 100%. In theory, a result greater than 100% should never happen. However, it can happen when the resolution of the the numerator is not the same as the resolution of the denominator. For example, the paging time is expressed in units of 1024 microseconds.

One goal of the Metering Interface is to give the user the most complete and undoctored information possible. That is, the Metering Interface chooses not to round the percentage down to 100%, but rather to give the actual data to the user, who can then interpret them.

When a ratio greater than 100% occurs, the bar graph displays are filled in with a darker stipple.

Metering Overhead When :Within Functions is Used

When you use the :Within function keyword to Meter Form or Meter in Process, the metered functions are encapsulated, and the encapsulations show up in the metering results. You will see functions that are obviously part of the metering facility.

These encapsulations usually take only about 4-500 microseconds, so they are usually insignificant compared to the other data. With the default filtering, they are almost never visible in the display. However, when you are metering code that takes less than a couple of milliseconds, the overhead spent in these encapsulations becomes significant and they appear in the display. In a call tree they are usually top level nodes, and so you can easily ignore them by hoisting the real top level nodes of interest. In function call metering there is currently no way to eliminate these functions automatically.

Metering Results Are Not Usually Repeatable

Note that although you can repeat a metering run, the results themselves are usually not repeatable. For example, paging performance depends on what pages are currently in virtual memory, and this is constantly changing. The metering results depend on all kinds of events that might be occurring in Genera, such as sequence breaks, incremental garbage collection, notifications, network services, and other processes. (See the section "The Effects of Sequence Breaks on Metering Results",

page 66.) In addition, variations in user code itself, such as caching, often change the metering results from one run to the next.

There are techniques for looking below the surface of the metering results, to determine how reliable the results are. Sometimes it is useful to meter the same thing several times in several different ways. If some aspect of the data seems out of the ordinary or suspicious, you can look at a histogram to see whether all of the data points are clustered together, or whether a few data points are at one extreme. You can do this by expanding the displayed data, or (when available) adding the output subfields Dist or WDist. See the section "Expanding Metering Data", page 61.See the section "Distribution of Metering Data", page 61.

Expanding Metering Data

Often the metering results display a summary of the collected data, and additional data is available to you. You can position the mouse over a piece of data, and click Middle to expand it.

For example, in a Function Call metering run, the column Inclusive Time shows the *total* amount of time spent inclusively in the function, which is a sum of the inclusive time for each call of the function. The function might have been called hundreds or thousands of times. Click Middle on one of those pieces of data to get more information on the Inclusive Time. You will see information such as:

Lowest data point
Highest data point
Average
Standard deviation
Histogram of the data points

Usually you picture a histogram as having the majority of the data points gathered around one main peak. However, sometimes the data points are gathered around more than one recognizable peak; there might be an underflow peak (below the main peak) and/or an overflow peak (above the main peak). When the data points are gathered around more than one peak, the histogram is *multi-modal.* For multi-modal histograms, the display shows more than one histogram, in order to focus on each of the peaks. Thus there is always one histogram showing the main peak, and there might be one or two more histograms, showing the underflow and overflow peaks, if any.

For a graphic example: See the section "Distribution of Metering Data", page 61.

Distribution of Metering Data

Some output fields collect information about the distribution of the data points. This information is available in the "Dist" and "WDist" output subfields, which are usually not part of the default display, but can be added with the Add Output Subfield command

The "Dist" output subfield stands for "Distribution" and "WDist" means "Weighted Distribution". Each shows a small graphic representation of the data points. The middle of the graph is the average; the left-hand edge is 0, and the right-hand

edge is twice the average. If there is data whose value is greater than twice the average, a gap appears afer the right-hand edge, and a smaller horizontal bar appears to its right; this represents the data whose values are greater than twice the average.

Dist — The height of each bar is related to the call count. That is, for the inclusive time output field, if several calls to a function fall within the same range of time, the height of each bar is controlled by the number of function calls within that category.

WDist — The height of each bar shows how much weight that data point contributed to the average. That is, for the inclusive time output field, several calls to a function might fall into the same range of time; the height of that bar is controlled both by the product of the number of calls in that range, and the (average) amount of time the calls in that range took.

Below we generate a metering run that has a wide distribution of data, show how the Dist and WDist fields appear on the screen, and discuss their significance.

Generating the Metering Run

We generated these results by Meter Form, where the arguments were:

```
CALL-TREE Metering Run
Created: 2/10/88 17:40:12
Form: (LOOP FOR I DOWNFROM 1000 TO 0 DO (FROB (FLOOR I 40)))
What was metered: Everything
Count: 1
Process: Metering Interface 1
Without Interrupts: Yes
```

We defined **frob** as follows:

```
(defun frob (n)
  (if (plusp n)
      (let ((limit (random n)))
        (loop repeat limit))
      (two-point)))

(defun two-point ()
  (let ((n (random 2)))
    (if (oddp n)
        (loop repeat 10 doing (random 2)))))
```

The purpose of this metering run was to show a widespread distribution of data. The functions **frob** and **two-point** have no other purpose.

Visual Appearance of Dist and WDist Fields

Calls Count	Excl Time Total	Avg	/Incl	Dist	WDist	Function
1	26337	26337.00				1 | ↓ MI::|Metered-form41|
1001	26023	26.00				2 | | ↓ FROB
961	39619	41.23				3 | | | ↓ RANDOM
961	63038	65.60				4 | | | | • CLI::RANDOM-INTERNAL
961	10484	10.91				4 | | | ↓ CLI::TYPEP-STRUCTURE
961	11362	11.82				5 | | | | ↓ NAMED-STRUCTURE-P
961	12711	13.23				6 | | | | | ↓ NAMED-STRUCTURE-SYMBOL
961	8662	9.01				7 | | | | | | • ARRAY-HAS-LEADER-P
40	1444	36.10				3 | | ↓ TWO-POINT
240	10431	43.46				4 | | | ↓ RANDOM
240	15673	65.30				5 | | | | • CLI::RANDOM-INTERNAL
240	2475	10.31				5 | | | ↓ CLI::TYPEP-STRUCTURE
240	2680	11.17				6 | | | | ↓ NAMED-STRUCTURE-P
240	2900	12.08				7 | | | | | ↓ NAMED-STRUCTURE-SYMBOL
240	2052	8.55				8 | | | | | | • ARRAY-HAS-LEADER-P

Figure 1. Dist and WDist Metering Output Subfields

The leftmost edge of the horizontal bar means 0. The small tick that descends from the middle of the horizontal bar of each Dist and WDist entry marks where the middle, or the average value is. The rightmost edge of the horizontal bar is twice the average value.

In some cases you will notice that past the rightmost edge of the bar there is a gap followed by another, smaller horizontal bar. This gap indicates that there are data points whose value is greater than twice the average value. Figure 1 shows such gaps in all of the functions except for two. In some cases so few data points occur there that they don't show up as a vertical bar. Still, the fact that there is a bar beyond the right edge indicates that some amount of data is there.

Using Dist to Understand the Average

Notice that the first function was called once. Its inclusive time is 100% of the run. Since it was called exactly once, there is only one data point. With only one data point, the Dist field clearly shows that all the data (one value) is clustered at the middle of the average.

The other functions were all called many times, so there are many data points. Here the Dist field shows more useful information. Look at the Dist field for the function **two-point**. The data points are clustered around two different values. That is, there is a peak somewhere below the average and another peak above the average. There is no data at all appearing at the average. (This is entirely due to the definition of **two-point**.) Here the Dist tells you that the average was calculated based on two distinct behaviors. In cases such as these, probably the average itself is unimportant, but the average of each separate peak is important. You can get that information by expanding the data (clicking Middle on the row).
The expanded data in Figure 2 confirms what we learned from the Dist output field; the data points are clustered around two different areas. The histogram is multi-modal (with two modes). The average of the 40 calls was 36.1. However the average of 20 calls was 64.75, and the average of the other 20 calls was 7.45.

```
Description of Excl Time of "TWO-POINT" in Run 2/10/88 17:40:12
Low:       6 High:      68     Count:   40         Avg: 36.1 Std Dev: 28.727644

  This histogram is multi-nodal:
  Main mode:
  Low:      61 High:      68     Bucket-size:    1   Count:   20         Avg: 64.75 Std Dev: 1.5580436
  Bucket Count    Bucket Count    Bucket Count    Bucket Count
  61:    1        63:    3        65:    9        67:    2
  62:             64:    2        66:    2        68:    1

     Underflow:
     Low:       6 High:       9     Bucket-size:    1   Count:   20         Avg: 7.45 Std Dev: 0.8645888
     Bucket Count    Bucket Count    Bucket Count    Bucket Count
     6:     2        7:    10        8:     5        9:     3
```

Figure 2. Expanded Data for two-point

Using WDist to Understand the Effect of Data Past the Gap

The Dist field shows where all the data points occurred. You can think of this as a seesaw where the average is the fulcrum. The seesaw is balanced on the average. If there is a gap and another horizontal bar to the far right, then the seesaw is longer on that end. The one thing that the Dist field cannot show you is how much longer that side is.

The data appearing past the gap affects the average. Its effect is based on two things: the number of data points there, and their values. The Dist field shows the number of data points there, but gives you no information about what their values. In other words, you don't know how long the gap is, or how much longer that side of the seesaw is.

The WDist field shows you the weighted distribution. It gives you an idea of how great an effect on the average the data points have. When computing an average from many data points, a small number of data points that have a high value have great impact on the average. On the other hand, a large number of data points with low values have a small impact on the average.

Look at the Dist and WDist fields for **cli::typep-function**. The Dist field shows that the great majority of data occurred well below the average. There is a gap, so some amount of data happened beyond twice the average. Since there is no vertical bar beyond the gap, very few data points occurred there. However, the WDist shows that the weighted value of those few data points was very large. In other words, a very few data points occurred quite far past twice the average. The right side of the Dist seesaw is much longer than the left.

How Time Spent in Other Processes Affects Metering Results

This section describes the difference between the output fields labeled "Time" and "Process Time". It begins by giving some background information.

When you meter a form and do not use **without-interrupts**, the scheduler will probably cause other processes to run, interleaved with the process in which you are metering. This has different effects on the metering results, depending on the metering type.

When you do metering and any part of the function being metering uses **without-interrupts**, the metering code itself is also within the scope of that **without-interrupts**. Usually the metering code takes significantly longer than the

user code being metered; this has one side-effect you might notice. For a function that uses **without-interrupts**, scheduler preemptions are more likely to occur when you are running metering the function than when you run your function without metering it. This happens because of the way the scheduler works; whenever a **without-interrupts** is exited, the scheduler immediately checks to see if other processes are waiting to run, and if so, it preempts the current process. Since the **without-interrupts** surrounds both the user code and the metering code (which often takes significantly longer than your code), there is more time for other processes to be ready to run. Thus preemptions are likely to occur immediately upon exiting the **without-interrupts** form during metering.

For all metering types except for Statistical Function Call and Statistical Call Tree:

Although the functions running within these other processes are not metered, the time spent in those processes does appear in the metering results. You will often see the function **stack-group-resume** in the metering results; this function indicates the time spent in other processes.

(Note that the results on paging time, number of page faults, and consing do not have this problem; they are collected on a per-process basis.)

When using Meter Form for short runs, it is sometimes useful to provide the :Without Interrupts keyword as Yes, to execute and meter the form from start to finish without allowing the scheduler to give control to other processes. This prevents irrelevant data from being collected and displayed. Note that it can also be dangerous, because your program and the metering code runs without interruption; unless the program is short, this might take a very long time and use up a lot of space on the machine.

For Statistical Function Call and Statistical Call Tree metering types:

These types of metering collect two kinds of data on time. The column labeled Time includes all the time spent during the metering run; that is, both the time when the process of interest was running, and the time when other processes were running. This is the equivalent to the Time displayed in the other kinds of metering, as described above. However, the column labeled Process Time includes only the time during which the process of interest was running.

The Process Time output field is usually more valuable information than the Time output field. Another interesting piece of information is the subfield of Process Time which shows the proportion of time spent in the process with respect to time spent in the metering run. This gives you an idea of the proportion of computing power that was allotted to your process during the metering run. Note that Statistical Function Call collects data on exclusive time, so that subfield is called /Excl, and it is under the field Exclusive Process Time. In contrast, Statistical Call Tree collects data on inclusive time, so that subfield is called /Incl, and it is under the field Inclusive Process Time.

The Effects of Sequence Breaks on Metering Results

Symbolics computers periodically take a *sequence break*, an asynchronous interrupt. During sequence breaks some mouse-tracking, I/O interrupts, and disk events might occur. Also, during the sequence break control might switch to the scheduler, which then checks to see whether other processes are waiting to be activated. The value of the variable **si:*default-sequence-break-interval*** controls how many sequence breaks occur before the scheduler is activated.

Although sequence breaks can occur even during **without-interrupts**, control is never switched to the scheduler inside a **without-interrupts**. However, there is some amount of overhead due to the sequence break itself.

The metering tools cannot measure exactly the overhead due to sequence breaks, so that overhead shows up in the metering results. The effects of sequence breaks are quite easy to recognize in metering results. For example, if you meter a function hundreds of times, you would expect the inclusive times of the function for all class to be very similar. However, you might notice that one call takes 250 microseconds or so (this varies from one machine to another) longer than the other calls; this extra time is probably due to a sequence break.

Sequence breaks are not something you should try to control; they happen in the normal operation of the machine. However, it is good to be able to recognize a sequence break, so you won't be concerned when you notice this anomaly in the metering results.

The Effects of Paging on Metering Results

When you are metering something, the metering code and data significantly increase the working set of your program. Thus, paging occurs more frequently during metering than it would otherwise occur.

If a page fault occurs during the execution of a function, the effect on the metering results is very large. You might notice this in a Call Tree metering run; a function might be called ten times, and its inclusive time for nine of those calls is approximately equal, but the inclusive time spent in one of the calls is far greater. If you notice this, look in the PFs or PS output field for that function call; probably a page fault occurred during it.

Since the Function Call metering runs display data indexed by function, the output fields show the total time spent in each function, not the time for each individual call of a given function. If you notice that a page fault occurred during a Function Call metering run, you can usually observe the effect of paging by expanding a piece of data (by clicking Middle on it) in the Inclusive Time field. This shows a histogram of the inclusive time in each of the calls to that function. You might notice one data point at the high extreme, which is probably the function call during which the page fault occurred. See the section "Expanding Metering Data", page 61.

We mention this for general background. If your goal in metering is to reduce paging time, then the extreme data points that occur due to paging represent information that is directly helpful to your goal. However, the goal might be to in-

crease efficiency of a program in aspects other than paging. In that case, you would probably ignore the extreme data points caused by paging. If your goal is more general (simply to improve performance, however it can best be done), you would probably try to weigh the effects of paging to determine whether it is worthwhile to spend effort in reducing paging time. In this case the histogram would be useful because it probably gives you some idea how often page faults are occurring by the number of data points are at the high extreme.

Page Fault Output Fields

Several of the metering types have two output fields related to paging: PFs (number of page faults) and PS (paging system time).

Usually, the time spent in the paging system is more valuable information than the number of page faults. The performance of your program is affected by the time spent in the paging system; you can compare the proportion of time spent paging versus time spent in the function itself, to get an idea how significant the effects of paging are on the function.

The data under PS is measured in microseconds, and it is also quantized in units of 1024 microseconds. That is, if the display shows that a function spent 1024 microseconds in the paging system, in reality that amount of time could be anywhere between 1 and 1024 microseconds. When PS data is totalled, the effects of the quantizing are cummulative. This means the results become more accurate as the sample size increases.

It is possible to have zero page faults but still spend a small amount of time in the paging system. This might indicate a map miss (that is, the paging system experiences a miss in the hardware cache that changes a virtual address to a physical address; it is necessary to page in that cache) or some other background activity in the paging system.

Interpreting Results of Meter in Process

Metering is implemented by noting when a function is entered and when it is exited. When using Meter Form, the form is always executed from start to finish, and the metering data is collected in entirety.

In contrast, when you use Meter in Process, you stop and start the metering explicitly. This means there is the possibility of starting or stopping metering in the middle of the execution of a function being metered. This is not necessarily bad, but it does have an effect on the data.

Consider what happens if you are doing a Call Tree metering run and you start the metering in the middle of a call tree. The function at the top of the tree (the root) was entered before metering was started, so the metering tools cannot collect data starting at that level of the tree. Instead, the callees of that function are metered, and they appear to be roots in the metering results. (A root function has level 1 in a call tree.) Thus the callees of the real root function appear as disconnected roots in the display, because the real root function was not metered from the beginning of its execution.

Now consider what happens if you are doing a Function Call metering run and you stop the metering before a function completes its execution. If that function was called only once, its Call Count is zero; this informs you that it was entered but not exited. However, if the function was called more than once, data has already been collected and will appear in the total time spent in the function, and the other fields. In this case the Call Count will be some integer which represents how many times the function was executed completely (both entered and exited) during metering. In this situation there is no way of knowing that the function was entered one additional time without being exited.

In summary, the metering tools collect data only for functions that are both entered and exited during metering. Sometimes you can control the starting and stopping of Meter in Process runs carefully, with the goal of not starting or stopping in the middle. In other cases, the metering results contain the information you want, even if the data is incomplete.

The Effects of the Sample Size in Statistical Metering

For Statistical Function Call and Statistical Call Tree metering, the results are more valuable as the number of samples increases. For example, if you meter a function that runs so quickly that only one sample takes place, the metering results will not be at all representative of the function's performance.

We suggest that you describe a metering run (click Middle on a metering run) to find out how many times sampling occurred during the metering. In Statistical Function Call metering you can also find out how many times a particular function was sampled (that is, how many times that function was being executed during the sampling) by expanding a portion of data (click Left on a piece of data).

Note that Statistical Program Counter metering runs your function again and again, in order to achieve a representative sample. This is not done by Statistical Function Call or Statistical Call Tree metering.

Statistical Metering of Code That Uses without-interrupts

The metering types Statistical Function Call and Statistical Call Tree do not accept the keyword :Without Interrupts. These metering types work by sampling the function periodically. Sampling cannot take place when code is running inside a **without-interrupts** form, because the scheduler will not interrupt that code in order to allow the metering process to sample the code. For example, if you wrap a **without-interrupts** form around a function and then try to meter that function with either the Statistical Function Call or the Statistical Call Tree metering type, the result will be no data. This holds for any portion of the code which is within the scope of **without-interrupts**.

The Statistical Program Counter metering type is done at the microcode level, so it can meter code within the a **without-interrupts** form.

6.1.1.4. Controlling Metering Within Lisp Code

mi:with-metering-enabled &body *body* *Special Form*

Use this special form to specify where metering should be enabled. All code in the dynamic scope of the *body* will be metered when you use Meter Form or Meter in Process, and specify :Only When Enabled. Alternatively, you can create a metering run by using **mi:with-new-metering-run** instead of the commands in the Metering Interface.

For example, you might want to exclude from metering any code that does preliminary set-up work, or performs a transition from one state to another, or cleans up afterward. The following example enables metering for two portions of the code:

```
(mi:with-new-metering-run (:metering-type :call-tree)
  (setup-code)
  (mi:with-metering-enabled
    (first-step))
  (transition-code)
  (mi:with-metering-enabled
    (second-step))
  (cleanup-code))
```

mi:with-new-metering-run *((*&key *:metering-type :name :process :without-interrupts)* &body *body)* *Special Form*

Creates a new metering run without using the Metering Interface; note that you need to use the Metering Interface to view the results. Use this special form in conjunction with **mi:with-metering-enabled**. All code of the *body* is executed, and any code within a **mi:with-metering-enabled** form is metered. The result is a metering run, which is placed in the Metering History pane of the Metering Interface. This metering run is not necessarily current, so to display it you should click Left on the metering run.

:metering-type	One of the following: **:function-call**, **:call-tree**, **:page-fault**, **:statistical-function-call**, **:statistical-call-tree**. The default is **:function-call**. See the section "Choosing a Metering Type", page 48.
:name	A string used to identify this metering run. There is no default for *:name*.
:process	The process in which to execute and meter the body. The default is the Metering Interface process.
:without-interrupts	If **t**, the code within **mi:with-metering-enabled** is executed inside a **without-interrupts** form. This means that no other process can interrupt the execution of the metering run. This should be used with caution,

because it can be dangerous for any code that does a lot of consing or takes a long time. If **nil**, the body is executed normally, and the results may show time spent in other processes. (Note that the functions running in other processes are not shown, but the time spent in them is shown). The default is **nil**. If the metering type is **:statistical-function-call** or **:statistical-call-tree**, you should not supply *:without-interrupts* as **t** because no sampling would take place.See the section "Statistical Metering of Code That Uses **without-interrupts**", page 68.

See the special form **mi:with-metering-enabled**, page 69.

6.1.1.5. Dictionary of Commands in the Metering Interface

Add Output Field Command

Add Output Field *metering-run new-field before-field*

Adds a new field to the display of the metering run. The available fields depend on the type of metering.

metering-run	A metering run. You can click on a metering run in the Metering History.
new-field	A field of data not already being displayed.
before-field	States where to place the new field in the display; the new field is placed immediately to the left of the *before-field.*

You can do this by clicking c-m-Left on a metering run. After entering the field to be added, a bar will appear somewhere within the output field. You can move that bar horizontally until you have it where you want the subfield to be placed, and then click Left to add the field to that position.

This command is available only within the Metering Interface.

Add Output Subfield Command

Add Output Subfield *metering-run new-subfield before-field*

Adds a new subfield to the display of the metering run. The available subfields depend on the type of metering.

metering-run	A metering run. You can click on a metering run in the Metering History.
new-subfield	A subfield of data not already being displayed.
before-field	States where to place the new field in the display; the new field is placed immediately to the left of the *before-field.*

You can do this by clicking c-m-Left on an output subfield. After entering the subfield to be added, a bar will appear somewhere within the output field. You can move that bar horizontally until you have it where you want the subfield to be placed, and then click Left to add the subfield to that position.

This command is available only within the Metering Interface.

Dehoist Command

Dehoist *metering-run call-tree-node keyword*

After you have hoisted a node, you can use Dehoist to change the display to use a different root node which is no longer displayed. The default is to restore the display to use the previous root node.

keyword {:number of levels}

:number of levels An integer or "All the way". The default is the integer that would restore the display to its previous root. All the way means to restore the display to reinstate the top-level function as the root.

You can do this by clicking s-m-Left on a node that has been hoisted.

This command is available only within the Metering Interface, and only for call tree displays.

Delete Output Field Command

Delete Output Field *metering-run output-field*

Deletes an output field from the display of the metering run.

metering-run A metering run. You can click on a metering run in the Metering History.

output-field An output field, which is one of the column headers. You can type in the name of an output field, or click on one in the display.

You can do this by clicking sh-Middle on the output field you want to delete.

This command is available only within the Metering Interface.

Delete Output Subfield Command

Delete Output Subfield *metering-run output-subfield*

Deletes an output subfield from the display of the metering run.

metering-run A metering run. You can click on a metering run in the Metering History.

output-subfield An output subfield, which is one of the column sub-headers. You can type in the name of an output subfield, or click on one in the display.

You can do this by clicking sh-Middle on the output subfield you want to delete. You can achieve the same effect by positioning the mouse over the

This command is available only within the Metering Interface.

Delete Metering Run Command

Delete Metering Run *metering-run*

Deletes a metering run from the Metering History.

metering-run A metering run. You can click on a metering run in the Metering History.

You can do this by clicking sh-Middle on a metering run.

This command is available only within the Metering Interface.

Describe Metering Run Command

Describe Metering Run *metering-run*

Describes a metering run, including the date and time of the run, what code was metered, and the metering parameters that were used.

metering-run A metering run. You can click on a metering run in the Metering History.

You can do this by clicking Middle on a metering run.

This command is available only within the Metering Interface.

Describe Output Field Command

Describe Output Field *output-field*

Describes the meaning of the data displayed in the *output-field.*

output-field An output field or subfield, which is one of the column headers. You can type in the name of a field, or click on one in the display.

You can do this by clicking Middle on an output field.

This command is available only within the Metering Interface.

Expand Field Command

Expand Field *metering-run output-field function*

Expands the data identified by *output-field* (a column) and *function* (a row), for the given *metering-run*.

metering-run	A metering run. You can click on a metering run in the Metering History.
output-field	An output field, which is one of the column headers. You can type in the name of an output field, or click on on in the display. This identifies the column of interest.
function	The function spec of the function for which data should be expanded. You can type in the function spec, or click on one in the display. This identifies the row of interest.

You can do this by clicking Middle on the piece of data you want to expand.

This command is available only within the Metering Interface.

Help Command in Metering Interface

Help *command-name*

Displays the documentation about the Metering Interface command.

To get a list of the Metering Interface commands, press the HELP key.

You can do this by clicking Middle on a command that appears in the metering command menu.

Hide All But Path to This Node Command

Hide All But Path to This Node *call-tree-node*

Customizes a call tree metering display to show only the path to the given node, by removing functions from the display that do not lead directly to this node. This does not remove any descendants of this node from the display.

You can achieve the same effect by positioning the mouse over a node, clicking Right, and choosing this command. This command is available only within the Metering Interface, and only for call tree displays.

Hide Node Children Command

Hide Node Children *call-tree-node*

Removes all the children of a node from a call tree metering display.

You can do this by clicking sh-Left on a node which is partially or completely open.

This command is available only within the Metering Interface, and only for call tree displays.

Hoist Node Command

Hoist Node *metering-run call-tree-node*

Changes a call tree metering display to focus on a certain node as if it were the root node. Removes all functions from the display which are not descendants of this node. When you hoist a node, it is often useful to add the /Root subfield to one or more fields of interest. For example, the /Root subfield of Exclusive Time output field shows the fraction of exclusive time spent in a given function, with respect to the new root (as opposed to /Run, the fraction of time in a given function with respect to the whole run).

You can do this by clicking s-m-Left on a node.

This command is available only within the Metering Interface, and only for call tree displays.

Lock Results Display Command

Lock Results Display

This is useful when customizing the display of metering results. You will notice that normally when you add or remove output fields, the Metering Interface immediately updates the display. This can be cumbersome. When you know you will be adding or deleting one output field after another, you can use this command to prevent the updating of the display. After you have finished specifying what output fields should be displayed, use Unlock Results Display to update the display of metering results.

This command is available only within the Metering Interface.

Meter Form Command

Meter Form *form metering-type what-to-meter keywords*

Immediately executes and simultaneously meters the *form* and displays the results. This command is available only within the Metering Interface.

form — Any Lisp form

metering-type — {Function Call, Call Tree, Page Fault, Statistical Function Call, Statistical Call Tree, Statistical Program Counter.} See the section "Choosing a Metering Type", page 48.

what-to-meter — {Everything, Only when Enabled, Functions.}

Everything	Meter everything within the form/process.
Only when Enabled	Meter only the code which is surrounded by a **mi:with-metering-enabled** form.

Within Functions — Meter only within the functions specified. You will be prompted for :Metered functions, and you should enter the functions of interest. See the section "Metering Overhead When :Within Functions is Used", page 60.

keywords — The keywords allow you to specify parameters that control the metering run. The keywords vary according to the *metering-type*.

All types of metering accept these keywords:

:Count — {*integer*} Execute the form this many times, and collect data only on the last run of the code. The default is 1. Note that often the first time a form is executed is not a representative run, for a variety of reasons. For example, sometimes some compilation occurs during the first execution of a form. Another example is paging; probably significantly more paging is necessary the first time a form is executed than the subsequent times. Often using this keyword is useful for metering a more representative run.

:Name — {*name*} A name to be used when printing and describing this run. This name will appear in the Metering History window pane.

The following keyword is accepted by Function Call, Call Tree, Page Fault, and Statistical Program Counter:

:Without Interrupts

{Yes No} Yes executes the form inside a **without-interrupts** form. This means that no other process can interrupt the execution of the metering run. This should be used with caution, because it can be dangerous for any code that does a lot of consing or takes a long time. When No, the form is executed normally, and the results may show time spent in other processes. (Note that the functions running in other processes are not metered or displayed, but the time spent in them is shown). The default is No. Using :Without Interrupts is useful for preventing irrelevant data from being collected and displayed, but it does not usually make the environment more representative (unless the code is typically executed within a **without-interrupts** form).

Note that if you specify both :Only When Enabled and :Without Interrupts as Yes, only the code within the **mi:with-metering-enabled** form is surrounded by **without-interrupts**.

Using :Without Interrupts is particularly useful for the Statistical Program Counter metering type, because it usually yields more repeatable results. When doing metering by sampling (instead of metering constantly), the results are more valuable when all of the runs are similar. If you are metering a form which has very different results each time it is run, the results of metering by sampling will be only a rough approximation of the characteristics of all the sampled runs, and may not be a good approximation of any given run.

The following keyword is accepted by Page Fault:

:Initially Flush All Pages
{Yes No} If Yes, all pages are flushed from virtual memory prior to the metering run. The default is No. This is useful when trying to set up the metering to occur in an environment in which the virtual memory does not contain the pages of interest; this might be representative of the first time a form is executed.

The following keyword is accepted by Statistical Program Counter:

:Resolution Percentage
{*float*} The default is 0.5%. The resolution percentage controls how many buckets the metering interface "searches" (it will skip all buckets that take up less than the resolution percentage of the total), and consequently how many times it must repeat your form. The finer (or smaller) the resolution, the more times it will have to repeat your form in order to investigate more buckets. For more information: See the section "Overview of How Metering Works", page 58.

Meter in Process Command

Meter in Process *process metering-type what-to-meter keywords*

This command is useful when you want to meter some code that normally runs within a process. For example, you might want to meter a function that normally runs within Zmail. You don't want to use Meter Form, because that would execute and meter the function immediately; instead, you want the function to be metered whenever it is normally called. Meter in Process allows you to meter one or more functions within a process, without explicitly calling those functions.

This command offers greater control over when the metering is started and stopped than does the Meter Form command. By default, the metering starts immediately after you finish entering the Meter in Process command. To stop the metering, you should select the Metering Interface. Either press END or answer YES to the displayed question, which is "Do you want to stop metering now?"

This command is available only within the Metering Interface.

process The process in which to meter.

metering-type {Function Call, Call Tree, Page Fault, Statistical Function Call, Statistical Call Tree.} Note that you cannot use the Statistical Program Counter metering type with Meter in Process. See the section "Choosing a Metering Type", page 48.

what-to-meter {Everything, Only when Enabled, Functions.}

Everything	Meter everything within the process.
Only when Enabled	Meter only the code which is surrounded by a **mi:with-metering-enabled** form.
Within Functions	Meter only within the functions specified. You will be prompted for :Metered functions, and you should enter the functions of interest. See the section "Metering Overhead When :Within Functions is Used", page 60.

keywords The keywords allow you to specify parameters that control the metering run. The keywords vary according to the *metering-type*.

All types of metering accept these keywords:

:Name {*name*} A name to be used when printing and describing this run. This name will appear in the Metering History window pane.

:Only When Enabled
{Yes No} Yes means to meter only those portions of the code that occur within the dynamic scope of a **mi:with-metering-enabled** form. No means to meter the specified function specs or the whole process. The default is No.

:Start and stop {Until End Chosen, Function Keys} Specifies the way in which metering is started and stopped. The default is Until End Chosen.

Until End Chosen means that metering is started immediately after the command is entered, and it is stopped when the user presses END in the Metering Interface.

Function Keys means that metering is started when the user enters Function (and stopped when the user enters Function). This allows you asynchronous control over when metering is started and stopped. See below for information on how :Start and stop interacts with :Mode lock p.

:Mode lock p {Yes No} Specifies whether the MODE LOCK key controls whether metering is on or off. Yes means that metering is turned on only when the MODE LOCK key is depressed. No means that the MODE LOCK key is not used to start and stop metering. The default is No. Note that MODE LOCK does not give an asynchronous signal to start or stop metering; instead, it gives a synchronous signal. This means that it might take a moment for the Metering Interface to poll for the status of MODE LOCK, so its effect is not immediate. See below for information on how :Start and stop interacts with :Mode lock p.

The following keyword is accepted by Page Fault:

:Initially Flush All Pages {*yes no*} If Yes, all pages are flushed from virtual memory prior to the metering run. The default is No. This is useful when trying to set up the metering to occur in an environment in which the virtual memory does not contain the pages of interest; this might be representative of the first time a form is executed.

Interaction between :Mode lock p and :Start and stop

Usually when users specify :Mode lock p as Yes, they specify :Start and stop as Until End Chosen. That way you cause metering to occur by pressing MODE LOCK; you cause it to stop occurring by releasing MODE LOCK; and you finally end the metering run entirely and display the data by selecting the Metering Interface and pressing END.

If you specify :Start and Stop as Function Keys and :Mode lock p as Yes, then metering is started only when you have pressed Function (**and** the MODE LOCK key is depressed. In other words, each of the keywords states how metering is started, so you must meet both requirements in order to start the metering. You can stop metering from occurring by releasing the MODE LOCK key, and cause metering to start again by pressing MODE LOCK again; You finally end the metering run entirely and display the data by entering Function).

Move Output Field Command

Move Output Field metering-run output-field before-field

Moves the specified *output-field* to the left of *before-field* in the display of metering results.

You can do this by clicking c-m-Middle on an output field.

This command is available only within the Metering Interface.

Re-Meter Command

Re-Meter *metering-run*

Repeats a metering run, selecting the type of metering and the code to meter from the specified metering run. You can then change the metering parameters, or start metering with the same parameters.

metering-run — A metering run. You can click on a metering run in the Metering History.

You can do this by clicking on [Re Meter] in the Metering Interface menu, or by clicking s-Middle on a metering run.

This command is available only within the Metering Interface.

Set Default Output Fields for Type Command

Set Default Output Fields for Type *metering-run*

Sets the defaults for displaying future metering runs of a certain metering type to be the same as the output fields displayed for the given *metering-run*. Any metering runs you do from now on will use these defaults. You can cause an existing metering run to use these defaults for display by using Set Output Fields of Run from Defaults on that metering run.

metering-run — A metering run. You can click on a metering run in the Metering History.

This command is available only within the Metering Interface.

Set Display Options Command

Set Display Options *metering-run*

Enables you to specify how the data of a metering run should be displayed, including how the data should be sorted. The display options depend on the metering type of the run.

metering-run — A metering run. You can click on a metering run in the Metering History.

You can achieve the same effect by clicking on [Set Display Options] in the Metering Interface menu. This command is available only within the Metering Interface.

Set Indentation Depth Command

Set Indentation Depth *metering-run integer*

Specifies how many levels to display without indenting, when displaying a Call Tree metering run. The levels after *integer* are indented. This helps you customize

the display to focus on an area of interest in the call tree, which might be many levels deep in the tree.

metering-run	A metering run. You can click on a metering run in the Metering History.
integer	Number of levels not to indent in the display. Indenting starts at the level after *integer*.

You can do this by clicking s-m-Middle on a displayed node, to start indentation after that node.

This command is available only within the Metering Interface, and only for call tree displays.

Set Output Fields of Run From Defaults

Set Output Fields of Run from Defaults *metering-run*

Sets the output fields of the given *metering-run* to the defaults. When you next display the metering run, the output fields will be displayed according to the defaults for metering runs of this type. This is useful when you have changed the defaults and you want a metering run to use the new defaults.

metering-run	A metering run. You can click on a metering run in the Metering History.

This command is available only within the Metering Interface.

Show All Node Descendants Command

Show All Node Descendants *call-tree-node*

Adds all the descendants of a node to a call tree metering display.

You can do this by clicking s-Left on a node.

This command is available only within the Metering Interface.

Show Node Children Command

Show Node Children *call-tree-node*

Adds all the children of a node to a call tree metering display.

You can do this by clicking Left on a node with undisplayed children.

This command is available only within the Metering Interface.

Show Metering Run Command

Show Metering Run *metering-run keywords*

Displays the results of the metering run in the Metering Results window, or if :Output Destination is specified, sends the results to that destination.

metering-run A metering run. You can click on a metering run in the Metering History.

keywords {:Output Destination}

:Output Destination
{buffer window printer} Sends the metering results to the specified buffer, window, or printer. Neither of the other two usual output destinations (files and streams) are supported. However, you can send the output to a file by first sending it to a buffer and then saving that buffer to a file.

You can achieve the same effect by clicking on [Show Metering Run] in the Metering Interface menu, or by clicking Left on a metering run.

This command is available only within the Metering Interface.

Unlock Results Display Command

Unlock Results Display

Use this to update the display of metering results, after you have locked the display by using Lock Results Display. You can achieve the same effect by clicking on the phrase **Unlock Results Display** which appears in the Metering Results pane when the display is locked.

This command is available only within the Metering Interface.

6.1.1.6. Macros for Metering the Execution Time of Forms

Sometimes a programmer wants a simple measure of how long a Lisp form takes to execute. It might not be worthwhile setting up the Metering Interface if only a quick test is desired, or if the amount of data collected by the Metering Interface is not needed. Probably the first alternative to come to mind is the Common Lisp **time** function: See the function **time** in *Genera 7.1 Patch Notes*.

Often, **time** is not adequate for simple metering. Since the behavior of the form varies depending on the state of the machine, one sample isn't enough. To understand the behavior of a form, it is useful to execute the form many times, and to see a histogram of the values so you can see the effects of "noise", bimodal behavior, or extreme data points. For an example: See the section "Distribution of Metering Data", page 61.

Here we document several macros that give you more flexibility and accuracy in metering the time of short forms. They are similar to **time** in the respect that they take a single form and return some simple metering information. They are more precise and informative than **time** in the measurement of time itself, although they provide less information than **time** with regard to the storage system, sequence-breaks, and consing.

The metering macros address the problems with using **time**. They enable you to meter a form by executing it many times and computing the average execution time. They simultaneously measure the metering overhead, which gives you an indication of the accuracy of the results.

Here we summarize the metering macros:

metering:with-part-of-form-measured *(*&key *(:no-ints* **'t***) :verbose :values (:time-limit* **1***) :count-limit)* &body *form*
Executes the *form* many times, and meters the subform that is surrounded by **metering:form-to-measure.**

metering:with-form-measured *(*&key *(:no-ints* **'t***) :verbose :values (:time-limit* **1***) :count-limit)* &body *form*
Executes the *form* many times, and meters the whole form.

metering:define-metering-function *name args (*&key *(:no-ints* **t***) :verbose :values :count-limit :time-limit)* &body *form*
Returns a compiled function which can be used to meter the *form* more than once. Useful when you know in advance that you will be metering a form repeatedly.

metering:measure-time-of-form *(*&key *(:no-ints* **'t***) :verbose :values :time-limit :count-limit)* &body *form*
Has the same effect as **metering:define-metering-function** in that it uses a compiled function to meter the *form*, but instead of returning the metering function, it runs it once to meter the form. The metering function is not saved for further use.

Probably **metering:define-metering-function** is the most generally useful of the group. It enables you to meter the form more than once. However, if you want to execute a form and meter only a portion of it, use **metering:metering-with-part-of-form-measured.**

Here we document each of the metering macros:

metering:with-part-of-form-measured *(*&key *(:no-ints* **'t***) :verbose :values (:time-limit* **1***) :count-limit)* &body *form* *Macro*

Executes the *form* many times, and meters the subform that is surrounded by **metering:form-to-measure.** If you want to meter the whole form, the macro **metering:with-form-measured** is more convenient: See the macro **metering:with-form-measured**, page 83.

This tells you how many microseconds (on average) were needed to evaluate the form or subform. It also measures the overhead of the metering code.

By default, the average time and the average overhead are printed out in a mouse-sensitive way. You can click on these averages to display the histogram of values that were used to compute them.

The keywords *:time-limit* and *:count-limit* can be used to control how many times the form is evaluated. The *:verbose* and *:values* keywords control the output of this macro. See the section "Keyword Options for Metering Macros", page 84.

This form is most useful in compiled code. When they are used in interpreted code, the results are primarily a measurement of the interpreter, and not the form.

See the section "Output of the Metering Macros", page 85.

metering:with-form-measured *(*&key *(:no-ints* **'t***)* *Macro*
:verbose :values (:time-limit **1***) :count-limit)* &body
form

Executes the *form* many times, and meters the whole form. This is an abbreviation for the most common case of **metering:with-part-of-form-measured**, in which the entire *form* is metered. If you want to meter a subform within a form, use **metering:with-part-of-form-measured**. See the macro **metering:with-part-of-form-measured**, page 82.

This tells you how many microseconds (on average) were needed to evaluate the form or subform. It also measures the overhead of the metering code.

By default, the average time and the average overhead are printed out in a mouse-sensitive way. You can click on these averages to display the histogram of values that were used to compute them.

The keywords *:time-limit* and *:count-limit* can be used to control how many times the form is evaluated. The *:verbose* and *:values* keywords control the output of this macro. See the section "Keyword Options for Metering Macros", page 84.

This form is most useful in compiled code. When they are used in interpreted code, the results are primarily a measurement of the interpreter, and not the form. See the section "Output of the Metering Macros", page 85.

metering:define-metering-function *name args (*&key *(:no-ints* **t***)* *Macro*
:verbose :values :count-limit :time-limit) &body *form*

Returns a compiled function which can be used to meter the *form* more than once. This is useful when you know in advance that you will be metering a form repeatedly.

This is an abbreviation for the following form (where where *keywords1* and *keywords2* are constructed according to the rules explained below):

```
(compile (defun function-name (arglist . keywords1)
           (metering:with-form-measured (keywords2) form)))
```

Any keywords specified in **metering:define-metering-function** will not be accessible in the function *function-name*. Any keywords omitted from the keyword list in **metering:define-metering-function** will become part of the arglist of *function-name*.

The compiled function *function-name* can be used to execute the *form* many times, and meter it. It tells you how many microseconds (on average) were needed to evaluate the form or subform. It also measures the overhead of the metering code.

By default, the average time and the average overhead are printed out in a mouse-sensitive way. You can click on these averages to display the histogram of values that were used to compute them.

The keywords *:time-limit* and *:count-limit* can be used to control how many times the form is evaluated. The *:verbose* and *:values* keywords control the output of this macro. See the section "Keyword Options for Metering Macros", page 84. See the section "Output of the Metering Macros", page 85.

metering:measure-time-of-form *(*&key *(:no-ints* **'t***) :verbose :values :time-limit :count-limit)* &body *form* — *Macro*

Has the same effect as **metering:define-metering-function** in that it uses a compiled function to meter the *form*, but instead of returning the metering function, it runs it once to meter the form. The metering function is not saved for further use.

See the macro **metering:define-metering-function**, page 83. See the section "Keyword Options for Metering Macros", page 84. See the section "Output of the Metering Macros", page 85.

Keyword Options for Metering Macros

The metering macros accept the following keyword arguments:

:no-ints The default is **t**. A non-**nil** value causes the metering to be done inside a **without-interrupts**. If the form is exceptionally long, or if it relies on other processes to work correctly, then **:no-ints** should be **nil**. If you specify **:no-ints t**, and the length of the form times **:count-limit** (if specified) is greater than five minutes, you will be prompted to check if you really want to disable the scheduler for that long.

:time-limit
Value is an integer, expressing a number of seconds. The default is 1 second. This specifies that the form should be repeated until this many seconds of real-time have elapsed. This includes the amount of time spent recording the metering results. **:time-limit** and **:count-limit** are mutually exclusive keywords.

:count-limit
Value is an integer. This specifies that the form should be repeated this many times. **:time-limit** and **:count-limit** are mutually exclusive keywords.

:verbose The default is **nil**. A non-**nil** value causes the full histograms to be printed out instead of just the averages. This keyword is overridden by the **:values** keyword.

:values The default is **nil**. A non-**nil** value causes nothing to be printed out; the metering results are represented by three returned values. The first is the average time, the second is the histogram of the times for evaluation

of the form, and the third is the histogram for the overhead loop. You can get other information by using the histograms as described below.

Using the Histograms

The following functions can be done to the histograms returned when you give the **:values** option:

```
;; To display the results of a histogram
(metering:display-collector histogram stream)

;; Returns the average value of the histogram
(metering:average histogram)

;; Returns the total of the data in the histogram
(metering:total histogram)

;; Returns standard-deviation of the data
(metering:standard-deviation histogram)

;; Maps over the buckets in the histogram
(metering:map-over-histogram-buckets
  histogram #'(lambda (low high count)))
```

To display a number in such a way so that clicking Middle will expand the data into a full histogram, you must present the data with the **'metering:metering-results** presentation type. For example:

```
(dw:with-output-as-presentation (:object rainfall
                                 :type 'metering:metering-results)
  (format t "~&The average rainfall was ~,5F inches."
          (metering:average rainfall)))
```

Output of the Metering Macros

By default, the output displays two or three quantities.

Average time

The first quantity, "Average time", is the time a single execution of the body took to execute, averaged over some number of repetitions. The number of repetitions can be controlled by using either the **:count-limit** or **:time-limit** keyword options.

Average clock overhead

The second quantity, "Avg clock overhead", is the amount of time spent by identical metering code metering the empty loop. This is provided for calibration (you can subtract this time from the "Average time") and to provide some measure of the significance of the result (if the "Average time" is close to the value of "Avg clock overhead", the results are suspect.

Clock variation

The third quantity may or may not be present. It begins with the phrase "A second sampling of the clock ....". This is printed out when a second measurement of the empty loop does not agree with the first. This indicates that something is making it hard to get reproducible metering results. This can be caused by many things. Although it does not always mean you should repeat the metering, it does mean that you should look at the numbers produced on such a run a little more carefully than normal.

The decision whether to print out the third value is controlled by the variable **metering:*tolerable-clock-variation***. Its value is a number between 0 and 1, which represents a percentage. When the two numbers differ by more than this percentage then the third value is printed.

Histograms are available

By default, the average time and the average overhead are printed out in a mouse-sensitive way. You can click on these averages to display the histogram of values that were used to compute them.

Usually you picture a histogram as having the majority of the data points gathered around one main peak. However, sometimes the data points are gathered around more than one recognizable peak; there might be an underflow peak (below the main peak) and/or an overflow peak (above the main peak). When the data points are gathered around more than one peak, the histogram is *multi-modal.* For multi-modal histograms, the display shows more than one histogram, in order to focus on each of the peaks. Thus there is always one histogram showing the main peak, and there might be one or two more histograms, showing the underflow and overflow peaks, if any.

6.1.2. New Display Debugger in Genera 7.2

There is a new version of the Display Debugger in Genera 7.2. The old Display Debugger will be found in the SYS:UNSUPPORTED directory.

6.1.2.1. Using the Display Debugger

The Display Debugger is a version of the standard Debugger that uses its own multi-paned window. You enter the Display Debugger from the standard Debugger by typing c-m-W, or by using the :Window Debugger command.

Figure 4 shows the Display Debugger, entered while in the Concordia editor.
Overview

The Display Debugger divides the screen into eight panes, showing various aspects of the program environment at the time of the error. It displays the name of the activity you are debugging in the top-left pane.

When the Display Debugger is entered, the various panes contains the state of the erring frame. Most common operations are available directly from the mouse.

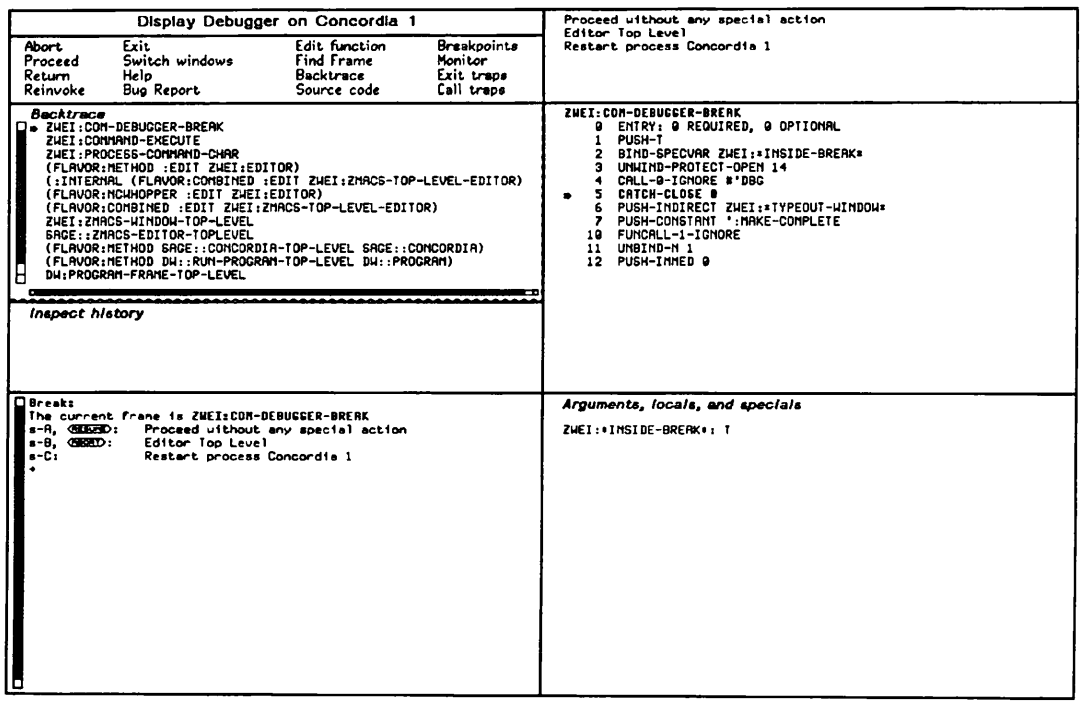


Figure 3. The Display Debugger

In addition to using the commands listed in the Command Menu in the Display Debugger, you can also use all of the usual Debugger commands by typing them on the keyboard.

Display Debugger Panes

Here is a description of each pane in the Display Debugger:

The Proceed Options pane -- the top righthand pane

The proceed options displayed in this pane are the same as seen in usual Debugger. Clicking [Left] on a proceed option takes that option. See the section "Using Debugger Proceed and Restart Options" in *Program Development Utilities.*

The Code pane -- the middle righthand pane

This pane displays the source code of the erring frame. If the Debugger can't find the source code, it displays the disassembled code instead.

If the you compiled the code with source locators, then the form which caused the Debugger to be entered is highlighted in boldface. Furthermore, the forms in the source code are sensitive as pieces of Lisp code. This means that you can click [Left] on a form in the source code to evaluate it in the context of the erring frame, or you can click control-meta-[Left] on a form to set a breakpoint at that place in the code.

If the Display Debugger is showing disassembled code, the current PC is highlighted in boldface with an arrow. You can click [Left] on the names of local and special variables to see their values, or you can click control-meta-[Left] on a PC in the disassembly to set a breakpoint at that PC.

Arguments, locals and specials pane -- the bottom righthand pane

This pane shows you all of the arguments and locals for the current frame, as well as any special variables bound in the frame. You can describe an argument, local, or special by clicking [Middle] on it. You can inspect it (with the Inspector) by clicking shift-[Middle] on it. You can modify the arguments, locals, or specials by pressing clicking control-metat-[right] on it.

Title pane -- the top lefthand pane

The Title pane shows you the name of the activity which you are debugging.

Command menu pane -- the second lefthand pane

This pane contains mouse-sensitive commands to do various things such as change your activity or placement in the Display Debugger.

If you position your cursor on one of the command names, the mouse documentation line shows what happens if you click left, middle, or right on a particular command.

A brief description of each of the commands in this pane follows:

Abort	Leaves the Display Debugger and aborts from the error.
Proceed	Proceeds from the error using the Resume proceed handler.
Return	Returns from the current frame. If you click [Middle] on this command, the Display Debugger asks you which frame you wish to return from.
Reinvoke	Reinvokes the current frame. If you click [Middle] on this command, the Display Debugger asks you which frame you wish to reinvoke.
Exit	Exits from the Display Debugger back to the normal Debugger. This neither aborts nor proceeds from the error.
Switch Windows	Switch to the original window where the error occurred. Press FUNCTION-S to return to the Display Debugger.
Help	Shows a brief help display.
Bug Report	Allows you to create, edit, and mail a bug report.
Edit function	Edit the function for the current frame.
Find Frame	Search down from the current frame for one whose function name contains a specified substring. Clicking [Middle] on the command causes the last search to be executed again from the new starting place.
Backtrace	Click [Left] on this command to display an "ordinary" backtrace, that is, a backtrace which hides invisible frames. Click [Middle] on this command to display a backtrace which does not hide invisible frames.

Source Code	Click [Left] on this command to see the source cfode for the frames, if it is available. Click [Middle] on this command to see disassembled code for the frames.
Breakpoints	Click [Left] for a display of all the currently set breakpoints. Click [Middle] to clear all the breakpoints. Click [Right] to get a menu of various other breakpoint-related commands.
Monitor	Click [Left] for a display of all the currently monitored locations. Click [Right] for a menu of various other monitor-related commands.
Exit traps	Click [Left] to set trap-on-exit for the current frame. Click [Middle] to clear trap-on-exit for the current frame. Click [Right] for a menu of various other trap-on-exit commands.
Call traps	Click [Left] to set trap-on-call for the current frame. Click [Middle] to clear trap-on-call for the current frame. Click [Right] for a menu of various other trap-on-call commands.

Backtrace pane -- the third lefthand pane

This pane displays the backtrace for the current error. The current frame is indicated by an arrow on the left. Clicking [Left] on a frame in this pane causes the current frame to be set to that frame. Clicking [Middle] on a frame shows the arguments with which that frame was called. Clicking [Right] on a frame pops up a menu for all the operations on that frame, such as set or clear trap-on-exit, disassemble the function for the frame, edit the frame's function, reinvoke this frame, return from this frame, and so forth.

Inspect history pane -- the fourth lefthand pane

This pane keeps track of all of the "complex" Lisp objects which you have examined. You can reexamine objects in this pane by clicking [Middle] or shift-[Middle] on them.

Interactor pane -- the bottom lefthand pane

This pane is where interaction with the Display Debugger takes place. You can use all of the usual Debugger commands and accelerators in this pane. For a list of Debugger commands: See the section "Debugger Command Descriptions" in *Program Development Utilities.*

6.1.3. The Garbage Collector Now Has Progress Notes

By default both the dynamic and ephemeral GC display their progress when the current process is waiting for the garbage collector.

You can control the display of progress notes for the GC with the new function, **si:enable-gc-progress-notes**, which controls the display of GC progress notes:

si:enable-gc-progress-notes &key *:dynamic :ephemeral* *Function*

Controls whether progress notes are displayed for garbage collection, and what their priorities are in relation to other progress notes. It takes two

keyword arguments, **:dynamic** and **:ephemeral**, allowing you to control the display of progress notes for each type of garbage collection independently. The default for both dynamic and ephemeral GC is that progress notes are displayed when the current process is waiting for the GC to complete (**:foreground**). Omitting an argument leaves its value unchanged. Each argument can have one of the following values:

nil	Progress notes are never displayed for GC.
:foreground	Progress notes are displayed only when the current process is waiting for GC to complete. This is the default for both ephemeral and dynamic GC.
:background	Progress notes are displayed for GC regardless of the GC's effect on the current process.
:override	Like **:background**, but causes GC progress notes to override all other progress notes in the status line display. In **:foreground** and **:background** modes, any other progress note overrides a GC progress note.

GC progress notes show the current internal state of garbage collection, and give little indication of the Garbage Collector's effect on the current process.

See also **tv:*show-system-internal-progress-notes*** which controls progress notes for other internal processes.

Examples:

To suppress the display of progress notes for the ephemeral GC, and allow the dynamic GC to display progress notes when it is affecting the current process (the default state), you evaluate the following:

```
(si:enable-gc-progress-notes :ephemeral nil)
```

The following form suppresses the display of all progress notes for the GC:

```
(si:enable-gc-progress-notes :ephemeral nil :dynamic nil)
```

6.2. Improvements to Utilities in Genera 7.2

6.2.1. Improvements to the Document Examiner in Genera 7.2

A number of improvements have been made to the Document Examiner:

- The Document Examiner displays now use dynamic windows.
- All Document Examiner commands can be entered from a menu or to a command processor prompt.

- Both the viewer pane and the overview display can be scrolled vertically and horizontally.

- Overview displays now use a multiline display of topic names instead of truncating them.

- You can show an overview of a topic in a Lisp Listener and also in the editor. Formerly, overviews were available only in Document Examiner. At the Command Processor, use Show Overview or click Right on a mouse-sensitive documentation topic. Use m-X Show Overview in the editor.

- A new Show Candidates command replaces the former Find commands. Show Candidates uses a heuristic searching algorithm that is more likely to find what you're looking for.

- Documentation topics read elsewhere in the system using the Zmacs or CP command Show Documentation are displayed in the new Background Viewer instead of in the default viewer.

- Cross-references and topic names displayed in a Lisp Listener or editor are now mouse-sensitive. Click Left on a topic name to see its associated documentation.

- Find Table of Contents has been renamed. To be consistent with the rest of the software environment, it is now called Show Table of Contents.

6.2.1.1. Document Examiner Now Uses Dynamic Windows

The Document Examiner now uses dynamic windows.

You can scroll the windows backwards and forwards using SCROLL and m-SCROLL, or the scroll bars. You can also scroll horizontally, using the horizontal scroll bars.

You can yank text from the Document Examiner screen by pressing the CONTROL key and dragging the mouse to mark the text you wish to yank. The click c-RIGHT on the mouse and put the marked text on the kill ring. Now you can put the text in a file or mail message.

You can search for text in the Document Examiner using s-R to search up and s-S to search down.

For more information on using dynamic windows:See the section "Using Your Output History" in *Genera User's Guide*. The Document Examiner commands menu pane is now a command processor. Type the command name at the prompt or click on the command name, type your lookup request, and press RETURN. Do not press RETURN after the command name, as before. For example, type:

```
Show Documentation SPACE catch-error-restart RETURN
```

6.2.1.2. New Show Candidates Command in Document Examiner

Show Candidates replaces the three documentation Find commands: Find Whole Word Candidates, Find Any Candidates, and Find Initial Substring Candidates. The capabilities of these commands are still provided and are available on the keywords menu as the exact, substring, and initial substring keywords to Show Candidates. To display the keywords menu in the commands pane, press m-COMPLETE after typing your lookup request. Press END to execute the command, or ABORT to abort it.

Heuristic matching. In addition to the old capabilities, Show Candidates offers a new default search scheme called heuristic matching. The advantage of heuristic matching is that you no longer have to think about the form of the word to supply as a lookup request; the heuristic approach finds singulars and plurals, gerunds, negations, and so on.

In this strategy, the search is based on the stem of your lookup request.

Example: The stem of "move" is "mov". Show Candidates move hypothetically matches any topic names and keywords containing "move", "moves", "moved", "moving", *move-mumble*, ":move", and so on.

Note that with the exception of "un", syllable prefixes are not considered matches, because it was deemed that the resulting candidates were not generally useful. So, a heuristic search on "move" does not match any forms of "remove".

Heuristic matching nets a larger number of candidates than the other search strategies. For example, a search based on "move" as an initial substring would not find "moving", nor would it find "*move-mumble*" or ":move" because leading punctuation is treated as part of the word. Substring matching (the old Find Any Candidates) would not match "moving". An exact matching scheme (formerly Find Whole Word Candidates) yields only "move".

Multiple-word adjacency lookup requests. Like the old Find commands, Show Candidates accepts lookup requests of more than one word. The default for multiple-word requests is still logical-**and**, any-order matching.

Unlike the old Find commands, Show Candidates suppports *adjacency lookup*. This means that candidates contain all words in the lookup request, where those words appear next to each other and in the order in which you specify them. Adjacency lookup further restricts the scope of your search, independent of the method of searching selected.

Example: Heuristic, any-order matching on "mouse moving" matches the topic names and keywords containing anywhere within them ":mouse", "*mouse", "mouse", or "mousing" as well as "move", "moves", "moved", and "moving". The search yields four candidates:

(FLAVOR:METHOD :MOUSE-MOVES TV:ESSENTIAL-MOUSE)
Altering the FED Character Box
Mouse Documentation Line in Zmacs
Moving the Cursor with the Mouse

The candidate "Altering the FED Character Box" is found because it has keyword

indexes including "mouse" and "move", as seen in the overview of the topic:

Using the Mouse on the Character Box
Move Black Font Editor Menu Item

Heuristic adjacency matching on "mouse moving" finds only one candidate:

(FLAVOR:METHOD :MOUSE-MOVES TV:ESSENTIAL-MOUSE)

Adjacency lookup is available on the keywords menu via m-COMPLETE.

6.2.1.3. Background Viewer Added to Document Examiner in Genera 7.2

Formerly, when you looked up documentation in a Lisp Listener or in the editor, the topic was added to the end of the current Document Examiner viewer, and the topic name was added to the end of the list of bookmarks. Now, such documentation is read into a special viewer called the *Background viewer*.

6.2.2. Improvements to the Garbage Collector

A bug that could cause the Ephemeral GC to crash creating a bignum after extremely large amounts of garbage have been collected has been fixed.

si:reorder-memory and the Optimize World command have been improved somewhat in Release 7.2. In particular, the paging and runtime performance of programs which use flavors will be further improved. In addition, using **si:reorder-memory** and Optimize World should not increase the size of IDS files.

6.2.2.1. The Start GC Command Has Been Enhanced

Start GC Command

Start GC *keywords*

Controls the operation of the Garbage Collector. Start GC with no keywords turns on both dynamic and ephemeral garbage collection.

See also "GC Cleanups", GC-ON, GC-OFF, FULL-GC, and GC-IMMEDIATELY.

keywords	:Cleanup, :Dynamic, :Ephemeral, :Immediately
:Cleanup	{Yes, No, Ask} Whether or not to run GC Cleanups to attempt to free address space. The default is No. The mentioned default is Yes, which does the maximum cleanup possible. Ask queries you about each cleanup before performing it. Start GC :Cleanup does not perform a GC or alter the mode of the background GC. See the section "GC Cleanups", page 94.

```
Command: :Start GC :Cleanup (Yes, No, or Ask [default Y
GC Cleanup Tasks
  Reset all input histories?  Yes  No
  Reset all presentation histories?  Yes  No
  Reset all editor histories?  Yes  No
  Reset LISP-TOP-LEVEL variables such as '* and '+?  Ye
  Reset interactor output histories?  Yes  No
  Clear some resources?  Yes  No
<ABORT> aborts, <END> uses these values
```

:Dynamic — {Yes, No} Enables or disables the dynamic Level of incremental GC.

:Ephemeral — {Yes, No} Enables or disables the ephemeral Level of incremental GC.

:Immediately — {Yes, No, By-Area} Perform a complete garbage collection right now. The mentioned default is Yes. Specifying By-Area offers a list of areas that can be flipped individually. When your address space has shrunk to where there is not quite enough free space for a GC :Immediately to complete, By-Area suggests some areas to flip that will maximize the amount of space reclaimed without risking running out of space completely. Start GC :Immediately also offers to run GC Cleanups.

For more information about the process of Garbage Collection, see "Theory of Operation of the GC Facilities" and "Invoking the Garbage Collection Facilities".

GC Cleanups

GC Cleanups are functions that you can request to run to make garbage collections more successful. A typical GC cleanup releases pointers to objects that are not strictly necessary for continued operation of an application, only convenient. Once the pointers are released, a subsequent GC can reclaim those objects. It is not appropriate for GC cleanups to delete useful information, such as mail or editor buffers.

Predefined GC cleanups exist that remove most pointers from the Lisp System to user objects. For example, the output histories of dynamic interactor windows are cleared. (These keep pointers to objects for mouse sensitivity.)

GC Cleanups are run using the "Start GC Command" with the keyword :Cleanup.

```
Command: Start GC (keywords) :Cleanup (Yes, No, or Ask [default Yes]) ■
```

GC Cleanups are defined using **si:define-gc-cleanup**.

6.2.3. Improvements to the System Construction Tool

6.2.3.1. Changes to defsystem Options

There is a new syntax for the **:bug-reports** options to **defsystem** . For compatibility, the old syntax will continue to work until, at least, 8.0.

:bug-reports option for defsystem

Specifies the mailing list for bug reports for the system and the purpose of the bug mail. The system has a bug report template with the values specified to keywords in the **:bug-reports** option. All values must be strings. The acceptable keywords are:

:name	Specifies the name of the bug report template. This name is used in all menus of bug report categories and is also used in the bug report's prologue describing the state of the machine on which the report was created. The default is the pretty name of the system.
:mailing-list	The name of the mailing list to which the bug report will be sent. The mailing list name must be specified exactly (i.e., we don't add "Bug-" to the string you give here). The default is "Bug-*system-name*" where *system-name* is replaced by the actual system name.
:documentation	The documentation associated with this bug report template. Said documentation is visible in the wholine documentation area when a menu of bug report categories is displayed. The default for this option is Report problems in the *pretty-name* system. where *pretty-name* is, of course, the pretty name of the system.

For example,

```
(defsystem ip-domain-server
    (:pretty-name "IP Domain Name Server"
     :bug-reports (:mailing-list "Bug-Domains")
         ...)
  ...)
```

specifies that the bug report template for the IP-Domain-Server system is called "IP Domain Name Server", the bug reports are sent to the "Bug-Domains" list, and the documentation string for the template is "Report problems in the IP Domain Name Server system.".

There are some new options to **defsystem**:

:maintain-journals option for defsystem

Controls whether or not a system is journalled. The default is **t**, to maintain journal files. Using **:maintain-journals nil** makes the system unjournalled. An unjournalled system is not patchable and has no version number, so loading it always loads the .NEWEST version of the files in the system.

:patch-atom option for defsystem

Controls how the patch-files for a system are named. Usually, the patch-file names are derived from the short-name of a system. **:patch-atom** lets you override the short-name.

:version-mapping option for defsystem

Controls the component mapping for component systems. For example:

```
(((:compile :newest) :released)      ;compiling :NEWEST loads :RELEASED
 ((:* :keyword) :number)             ;keywords snapshot the number
 ((:* :number) :number))             ;ditto for numbers
```

There is a refinement of the options to module dependencies, **:serial-definitions**, which combines **:serial** and **:uses-definitions-from.**

6.2.3.2. Changes to Patching

There is a new CP command that allows you to see the patches that you have not loaded. See the section "Show Additional Patches Command" in *Genera Handbook.*

New Interface to Finish Patch

There is a new menu interface for the Finish Patch (m-X) command. When you do m-X Finish Patch, an AVV menu comes up, allowing you to see your patch comment, the modified source files (if any) associated with the patch, and the patch author.

You have the options for editing the patch comment immediately or when you press END, and for sending mail about the patch.

You also can specify dependencies for loading the patch.

sct:require-patch-level-for-patch &rest *system-major-minor-specs* *Function*

Enforces a patch's dependency on some particular patch level in another system or systems. It is used at the head of any patch file that requires a certain patch level in some other system to load or operate correctly. For example:

```
(sct:require-patch-level-for-patch '(system 357. 510.) '(tape 69. 10.))
```

If the patch level requirements are not all met, it blows out, indicating at what level all specified systems need to be. The RESUME option skips loading further patches for the present system. This is available via the Finish Patch menu.

There is a new facility to control the loading of problem patches.

Dangerous Patches

Occasionally you need to make a patch to a system that, in some circumstances, might damage the system. For example, it might make changes to very low level internal functions or initialize parts of the system. Loading such a patch into a

running system could unpleasantly affect the operation of the system. Such a patch is referred to as a *dangerous patch.* You can declare a patch dangerous by placing the form **sct:dangerous-patch** at the beginning of the patch file.

sct:dangerous-patch *format-string* &rest *format-args* *Function*

Specifies a patch as being problematic to load (a *dangerous patch*) in some circumstances.

To declare a patch a dangerous patch, place a form containing **sct:dangerous-patch** at the beginning of the patch file, before the contents of the patch, to test for the conditions under which the patch should not be loaded.

For example, if you have a program that creates a list called ***my-results*** to store its results, you would not want to load a patch that reinitializes that list if the program was running. You should put a form like this at the beginning of the patch file:

```
(when (listp *my-results*)
  sct:dangerous-patch "This patch cannot be loaded because it
reinitializes *my-results*")
```

When you attempt to load the patch, **load-patches** checks the value of **sct:*dangerous-patch-action*** to determine the action to take.

Using **sct:dangerous-patch** at top level (not inside a conditional form) produces an error when you attempt to finish the patch.

sct:with-dangerous-patch-action *(action)* &body *body* *Function*

Allows you to bind **sct:*dangerous-patch-action*** during a **load-patches** operation. This is useful if you are loading patches under program control.

```
(sct:with-dangerous-patch-action :load (load-patches))
```

The possible values for **sct:*dangerous-patch-action*** are:

:skip	The default. Skips loading patches for the system.
:query	Queries you, allowing you to skip loading patches for the system or load the dangerous patch.
:load	Loads the patch inspite of its dangerous status.

The Load Patches CP command takes a keyword argument, :Dangerous Patch Action, that is the same as **sct:with-dangerous-patch-action.**

sct:*dangerous-patch-action* *Variable*

Controls the action taken when a dangerous patch is encountered in loading patches for a system. See the function **sct:dangerous-patch**, page 97.

:skip	The default. Skips loading patches for the system.
:query	Queries you, allowing you to skip loading patches for the system or load the dangerous patch.

:load Loads the patch inspite of its dangerous status.

6.2.3.3. Change in What Version of Component Systems Are Loaded When a System is Loaded

Now when you load a system that has component systems, SCT loads the versions of the component systems that were used to compile the system. The previous behavior was that if you loaded, for example, the Released version of a system, the Released versions of the components were also loaded. The new behavior is more consistent.

The default *component map* is now this:

```
(((:compile :newest) :released)      ;compiling :NEWEST loads :RELEASED
 ((:* :keyword) :number)             ;keywords snapshot the number
 ((:* :number) :number))             ;ditto for numbers
```

To get the old behavior, use this as the component map in the parent system (specified with the **:version-mapping** option to **defsystem**):

```
(((:compile :newest) :released)      ;compiling :NEWEST loads :RELEASED
 ((:* :keyword) :keyword)            ;any other keyword passes through
 ((:* :number) :number))             ;number maps to snapshotted number
```

The Load System CP command can also be used to override the new behavior. To load the Released version of *system-name*, and load the Released version of its components instead of the versions snapshotted by the compilation of *system-name*, you would use the following command:

```
Load System system-name :Version Released :Component Version Released
```

6.2.4. Improvements to the Compiler

6.2.4.1. Changes to Compiled Function Constants

In earlier releases, compiled function constants were localized with functions which called them only by the Optimize World command. This had a number of undesirable side effects, such as increased working set, and increased size of IDS worlds. In 7.2, constants are localized with functions when they are created, either by the compiler or by the **bin** file loader.

One consequence of the above change is that some minor restrictions which were previously only imposed by the **bin** dumper and Optimize World, are now imposed always. In particular, circular structures can not be used as compiled function constants, and code can not rely on constants being **eq**.

Completely compatible behavior with 7.1 can be achieved using the variable **si:*compiled-function-constant-mode***.

Example 1:

```
(defun foo () '#1=(#1#))
```

In 7.1, this function could be compiled from an editor buffer, but it could not be saved in **bin** files due to the circular constant. Additionally, Optimize World would recurse infinitely trying to copy the constant.

In 7.2, this function can never be compiled, since the code which localizes the constant with the function will recurse infinitely. See the variable **si:*compiled-function-constant-mode***, page 99.

Example 2:

```
(eval-when (compile load eval)
  (defvar *foo* (list 1 2 3)))

(defun foo ()
  (eq (cdr '#,*foo*) '#,(cdr *foo*)))
```

In 7.1, **(foo)** would return **t** until the world was optimized, after which it would return **nil** . In 7.2, since the list constants are copied afterthe load-time-eval, this function always returns **nil**. **si:*compiled-function-constant-mode*** *Variable*

Controls how constants are localized with compiled functions in normal compiled-function creation. Its value can be one of the following:

:share	This is the default. Compiled function constants are copied and shared to be immediately after the compiled function in memory.
:copy	Compiled function constants are copied to be immediately following the compiled function in memory. No attempt is made to share constants. In some cases this may result in faster loading of compiled functions and a larger working set for the resulting functions.
:unlocalized	Compiled function constants are not copied. Thus circular structures and EQ-ness of constants are preserved. However, the working set of running functions loaded in this manner is guaranteed to be larger, since the constants are guaranteed to be on separate pages. Additionally, garbage collection overhead wil be higher for dynamic constants, and IDS files may be larger.

Note that the only constants which are currently copied are lists, numbers, strings, and simple arrays.

Copying of compiled functions and this variable may be changed in a future release.

6.2.4.2. New Compiler Optimization

(funcall #'foo ...) now optimizes into **(foo ...)**. This improves performance but means that you can no longer use **funcall** to avoid inline compilation. The correct way to prevent inline compilation is to use the **notinline** declaration.

6.2.4.3. Changes to Cdr-Coding Behavior of Some Functions

The behavior of **rplacd**, **zl:nreverse**, **nreverse**, and **nreconc** has changed with respect to cdr-coding.

rplacd now returns a cdr-coded list in the form **(without-interrupts (rplacd (list 'a) (list 'b))**. It usually returns a cdr-coded list when interrupts are enabled.

zl:nreverse, **nreverse**, and **nreconc** now perform much more efficiently when passed cdr-coded lists. Usually, **rplacd**-forwarding no longer occurs in these functions.

The exact list destruction which occurs when calling **zl:nreverse**, **nreverse**, and **nreconc** is now undefined. In previous releases, it was the case that **(eq (last list) (nreverse list))** returned **t**. Now this may or may not be the case, depending on the cdr-coding of the list and the machine type.

6.2.4.4. Change to throw

The system now enters the debugger in situations such as

```
(CATCH 'A
  (CATCH 'B
    (UNWIND-PROTECT
        (THROW 'A 1)
      (THROW 'B 2))))
```

rather than sometings returning 2 and sometimes destroying your environment.

6.2.4.5. New Sub-primitive

There is a new sub-primitive for implementation of low-level things. It is used by the new implementation of **throw** **sys:%multiple-value-pop** *Function*

A function of 0 arguments. It pops a multiple group off of the stack and returns it as its values.

6.2.4.6. Bug in Read-Time Conditionalizations Fixed

Constructs such as `#+system::cmu` or `#+system:alcatraz` now work. Previously they signalled a package error.

6.3. Incompatible Changes to Utilities in Genera 7.2

6.3.1. Incompatible Change to sys:meter-function-entry and sys:meter-function-exit

Genera 7.2 includes an incompatible change to two undocumented functions. We document this change in case users have been redefining these functions to implement their own metering facilities. This incompatible change is a system improvement that prevents a race condition during trap-handling; the benefit is that metering results are more accurate.

In 7.2, both **sys:meter-function-entry** and **sys:meter-function-exit** take an argument. Prior to 7.2 they did not take an argument. This argument is the old value

of **sys:inhibit-scheduling-flag**. If you redefine these functions, they must both accept the new argument. Also, the bodies must include:

```
(setf si:inhibit-scheduling-flag old-inhibit-scheduling-flag)
```

6.3.2. Document Examiner Find Commands Replaced

The Find Commands in the Document Examiner have been replaced by a single Show Candidates command. See the section "Improvements to the Document Examiner in Genera 7.2", page 90.

7. Changes to the Debugger in Genera 7.2

7.1. New Features in the Debugger in Genera 7.2

7.1.1. New Features for Advice

To *advise* a function is to tell a function to do something extra in addition to its actual definition. Advising is achieved by means of the function **advise**. The something extra is called a piece of advice, and it can be done before, after, or around the definition itself.

The advise feature has been enhanced in Genera 7.2. Here are some new functions and variables: **si:advise-permanently** *function class name position* *Function*
&body *forms*

This form is identical to **advise**, except that forms advised by **si:advise-permanently** cannot be removed by **unadvise**. They must be removed by **si:unadvise-permanent**. See the function **si:unadvise-permanent** in *Program Development Utilities*.

function Specifies the function to put the advice on. It is usually a symbol, but any function spec is allowed. (See the section "Function Specs" in *Symbolics Common Lisp -- Language Concepts*.)

class Specifies either **:before**, **:after**, or **:around**, and says when to execute the advice (before, after, or around the execution of the definition of the function). For more information about the meaning of **:around**, **:before**, and **:after** advice:See the section "**:around** Advice" in *Program Development Utilities*.

name Specifies an arbitrary symbol that is remembered as the name of this particular piece of advice. It is used to keep track of multiple pieces of advice on the same function. If you have no name in mind, use **nil**; then we say the piece of advice is anonymous.

A given function and class can have any number of pieces of anonymous advice, but it can have only one piece of named advice for any one name. If you try to define a second one, it replaces the first.

Advice for testing purposes is usually anonymous. Advice used for customizing someone else's program should usually be named so that multiple customizations to one function have separate names. Then, if you reload a customization that is already loaded, it does not get put on twice.

position Specifies where to put this piece of advice in relation to others of the same class already present on the same function.

Position can have these values:

- *position* can be **nil**. The new advice goes in the default position: it usually goes at the beginning (where it is executed before the other advice), but if it is replacing another piece of advice with the same name, it goes in the same place that the old piece of advice was in.
- *position* can be a number, which is the number of pieces of advice of the same class to precede this one. For example, 0 means at the beginning; a very large number means at the end.
- *position* can have the name of an existing piece of advice of the same class on the same function; the new advice is inserted before that one.

forms Specifies the advice; they get evaluated when the function is called.

si:interpret-advice *Function*

Function must be a function spec of a compiled function that is currently advised. This specification is "sticky" until the next time all advice is removed from *function*. Until then, all advice for *function* is interpreted.

si:advise-permanently *function class name position* &body *forms* *Function*

This form is identical to **advise**, except that forms advised by **si:advise-permanently** cannot be removed by **unadvise**. They must be removed by **si:unadvise-permanent**. See the function **si:unadvise-permanent** in *Program Development Utilities*.

function Specifies the function to put the advice on. It is usually a symbol, but any function spec is allowed. (See the section "Function Specs" in *Symbolics Common Lisp -- Language Concepts*.)

class Specifies either **:before**, **:after**, or **:around**, and says when to execute the advice (before, after, or around the execution of the definition of the function). For more information about the meaning of **:around**, **:before**, and **:after** advice:See the section "**:around** Advice" in *Program Development Utilities*.

name Specifies an arbitrary symbol that is remembered as the name of this particular piece of advice. It is used to keep track of multiple pieces of advice on the same function. If you have no name in mind, use **nil**; then we say the piece of advice is anonymous.

A given function and class can have any number of pieces of anonymous advice, but it can have only one piece of named

advice for any one name. If you try to define a second one, it replaces the first.

Advice for testing purposes is usually anonymous. Advice used for customizing someone else's program should usually be named so that multiple customizations to one function have separate names. Then, if you reload a customization that is already loaded, it does not get put on twice.

position Specifies where to put this piece of advice in relation to others of the same class already present on the same function.

Position can have these values:

- *position* can be **nil**. The new advice goes in the default position: it usually goes at the beginning (where it is executed before the other advice), but if it is replacing another piece of advice with the same name, it goes in the same place that the old piece of advice was in.
- *position* can be a number, which is the number of pieces of advice of the same class to precede this one. For example, 0 means at the beginning; a very large number means at the end.
- *position* can have the name of an existing piece of advice of the same class on the same function; the new advice is inserted before that one.

forms Specifies the advice; they get evaluated when the function is called.

si:show-permanent-advice *Function*

Displays all functions which currently have permanent advice.

7.1.1.1. Compiled Advice

Prior to Genera 7.2, all advice was interpreted by default. Now, the you have the option of whether or not to compile or interpret advice. You can controll this globally, by using **si:*advice-compiled-by-default***, or individually (on a function by function basis) by using **si:compile-advice** and **si:interpret-advice**.

si:*advice-compiled-by-default* *Variable*

When this varible is set to **t**, **advise** and **si:advise-permanently** cause the advice to be compiled. When **si:*advice-compiled-by-default*** is set to **nil**, the advice is interpreted.

si:compile-advice *function* *Function*

Function must be a function spec of a compiled function that is currently advised. This specification is "sticky" until the next time all advice is removed from *function*. Until then, all advice for *function* is compiled.

si:interpret-advice *Function*

Function must be a function spec of a compiled function that is currently advised. This specification is "sticky" until the next time all advice is removed from *function*. Until then, all advice for *function* is interpreted.

7.1.2. New Display Debugger Interface

There is a new Display Debugger interface in Genera 7.2.

7.1.2.1. Using the Display Debugger

The Display Debugger is a version of the standard Debugger that uses its own multi-paned window. You enter the Display Debugger from the standard Debugger by typing c-m-W, or by using the :Window Debugger command.

Figure 4 shows the Display Debugger, entered while in the Concordia editor.

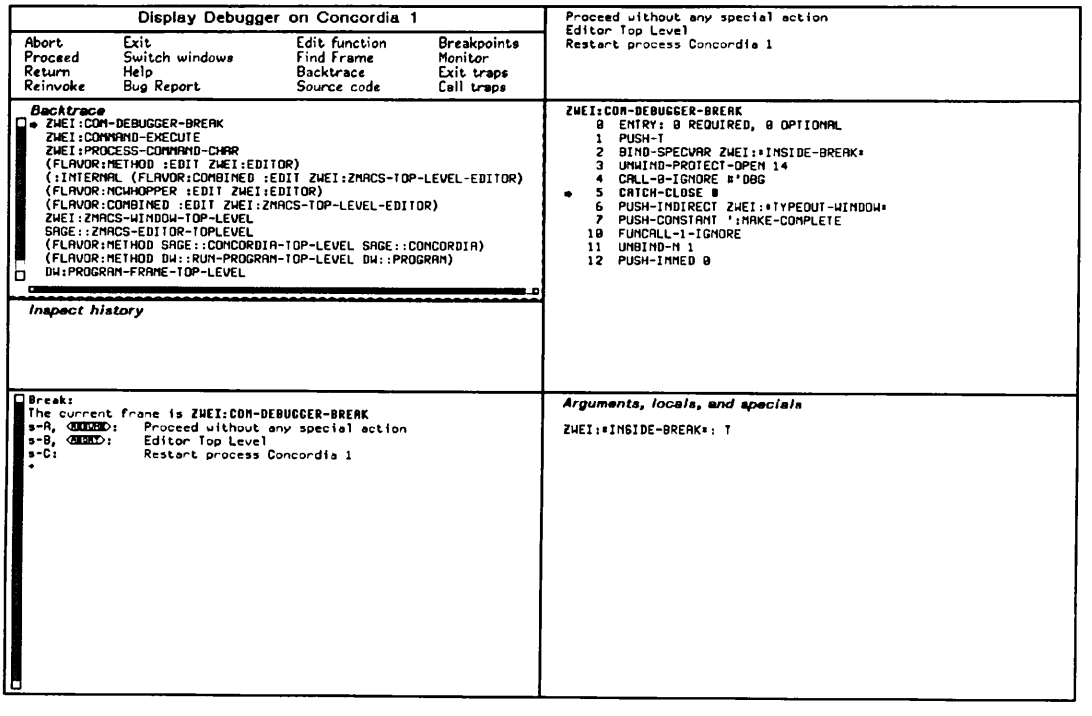


Figure 4. The Display Debugger

Overview

The Display Debugger divides the screen into eight panes, showing various aspects of the program environment at the time of the error. It displays the name of the activity you are debugging in the top-left pane.

When the Display Debugger is entered, the various panes contains the state of the erring frame. Most common operations are available directly from the mouse.

In addition to using the commands listed in the Command Menu in the Display Debugger, you can also use all of the usual Debugger commands by typing them on the keyboard.

Display Debugger Panes

Here is a description of each pane in the Display Debugger:

The Proceed Options pane -- the top righthand pane

The proceed options displayed in this pane are the same as seen in usual Debugger. Clicking [Left] on a proceed option takes that option. See the section "Using Debugger Proceed and Restart Options" in *Program Development Utilities.*

The Code pane -- the middle righthand pane

This pane displays the source code of the erring frame. If the Debugger can't find the source code, it displays the disassembled code instead.

If the you compiled the code with source locators, then the form which caused the Debugger to be entered is highlighted in boldface. Furthermore, the forms in the source code are sensitive as pieces of Lisp code. This means that you can click [Left] on a form in the source code to evaluate it in the context of the erring frame, or you can click control-meta-[Left] on a form to set a breakpoint at that place in the code.

If the Display Debugger is showing disassembled code, the current PC is highlighted in boldface with an arrow. You can click [Left] on the names of local and special variables to see their values, or you can click control-meta-[Left] on a PC in the disassembly to set a breakpoint at that PC.

Arguments, locals and specials pane -- the bottom righthand pane

This pane shows you all of the arguments and locals for the current frame, as well as any special variables bound in the frame. You can describe an argument, local, or special by clicking [Middle] on it. You can inspect it (with the Inspector) by clicking shift-[Middle] on it. You can modify the arguments, locals, or specials by clicking control-meta-[right] on it.

Title pane -- the top lefthand pane

The Title pane shows you the name of the activity which you are debugging.

Command menu pane -- the second lefthand pane

This pane contains mouse-sensitive commands to do various things such as change your activity or placement in the Display Debugger.

If you position your cursor on one of the command names, the mouse documentation line shows what happens if you click left, middle, or right on a particular command.

A brief description of each of the commands in this pane follows:

Abort	Leaves the Display Debugger and aborts from the error.
Proceed	Proceeds from the error using the Resume proceed handler.
Return	Returns from the current frame. If you click [Middle] on this command, the Display Debugger asks you which frame you wish to return from.

Reinvoke	Reinvokes the current frame. If you click [Middle] on this command, the Display Debugger asks you which frame you wish to reinvoke.
Exit	Exits from the Display Debugger back to the normal Debugger. This neither aborts nor proceeds from the error.
Switch Windows	Switch to the original window where the error occurred. Press FUNCTION-S to return to the Display Debugger.
Help	Shows a brief help display.
Bug Report	Allows you to create, edit, and mail a bug report.
Edit function	Edit the function for the current frame.
Find Frame	Search down from the current frame for one whose function name contains a specified substring. Clicking [Middle] on the command causes the last search to be executed again from the new starting place.
Backtrace	Click [Left] on this command to display an "ordinary" backtrace, that is, a backtrace which hides invisible frames. Click [Middle] on this command to display a backtrace which does not hide invisible frames.
Source Code	Click [Left] on this command to see the source cfode for the frames, if it is available. Click [Middle] on this command to see disassembled code for the frames.
Breakpoints	Click [Left] for a display of all the currently set breakpoints. Click [Middle] to clear all the breakpoints. Click [Right] to get a menu of various other breakpoint-related commands.
Monitor	Click [Left] for a display of all the currently monitored locations. Click [Right] for a menu of various other monitor-related commands.
Exit traps	Click [Left] to set trap-on-exit for the current frame. Click [Middle] to clear trap-on-exit for the current frame. Click [Right] for a menu of various other trap-on-exit commands.
Call traps	Click [Left] to set trap-on-call for the current frame. Click [Middle] to clear trap-on-call for the current frame. Click [Right] for a menu of various other trap-on-call commands.

Backtrace pane -- the third lefthand pane

This pane displays the backtrace for the current error. The current frame is indicated by an arrow on the left. Clicking [Left] on a frame in this pane causes the current frame to be set to that frame. Clicking [Middle] on a frame shows the arguments with which that frame was called. Clicking [Right] on a frame pops up a menu for all the operations on that frame, such as set or clear trap-on-exit, disassemble the function for the frame, edit the frame's function, reinvoke this frame, return from this frame, and so forth.

Inspect history pane -- the fourth lefthand pane

This pane keeps track of all of the "complex" Lisp objects which you have examined. You can reexamine objects in this pane by clicking [Middle] or shift-[Middle] on them.

Interactor pane -- the bottom lefthand pane

This pane is where interaction with the Display Debugger takes place. You can use all of the usual Debugger commands and accelerators in this pane. For a list of Debugger commands: See the section "Debugger Command Descriptions" in *Program Development Utilities*.

7.1.3. New Debugger Proceed Menu in Genera 7.2

Debugger pop up proceed menus are an alternative interface to the regular Debugger. When an error occurs, a menu pops up, enabling the user to select a retry option. Debugger proceed menus are perfect for situations where there is an abormal but expected error with a possible clean recovery. An example is programs that perform long file operations where there is a good possibility of a network break.

The following is a list of standard errors in the Genera system that causes a Debugger proceed menu to appear in case of an error:

fs:file-operation-failure
fs:unknown-pathname-host
fs:host-not-accessible-for-file
fs:host-not-available
sys:host-not-responding
sys:unknown-host-name
tape:mount-error

If you want to disable Debugger menus from appearing, set or bind the variable **dbg:*disable-menu-proceeding*** to **t**.

dbg:*disable-menu-proceeding* *Variable*

Use this to disable Debugger menu proceeding. When set or bound to **t**, it forces all error conditions to enter the standard Debugger.

If you have a condition that you want to add to the list of standard errors that causes a pop up Debugger menu to appear, use the macro **dbg:with-extra-debugger-menu-conditions**.

dbg:with-extra-debugger-menu-conditions *(conditions)* &body *body* *Macro*

Given a set of *conditions*, executes the *body* with the *conditions* added to the standard list of conditions which cause a Debugger menu to appear, rather than entering the Debugger.

This macro is useful for error conditions which are abnormal but expected, and for which there is a possible clean recovery. For example, if you have a program that is performing long file operations, and you expect that it may run into a network break, you can use

dbg:with-extra-debugger-menu-conditions to provide a pop-up menu of options when it runs into the network break, rather than putting you into the standard Debugger.

conditions is a list of flavors.

body is the code that may encounter the conditions.

Here is an example of **dbg:with-extra-debugger-menu-conditions** used with code designed to save the results of a computation on a file. **dbg:with-extra-debugger-menu-conditions** provides for a Debugger menu interface in case any kind of file or network error occurs.

```
(dbg:with-extra-debugger-menu-conditions (fs:file-error sys:network-error)
  (catch-error-restart-with-form ((fs:file-error sys:network-error)
                                  "Skip saving file ~A." name)
      (abort-current-command)
    (error-restart ((fs:file-error sys:network-error)
                    "Retry saving file ~A." name)
      (with-open-file (stream name :direction :output)
        ... <code to output data to the file> ... )
      )))
```

To disable the Debugger menu from appearing, you can set or bind the variable **dbg:*disable-menu-proceeding*** to **t**.

Figure 5 shows an example of a Debugger menu which occurred when a user tried to save a file on a disk when there was not enough room. This menu is unrelated to the code example above.

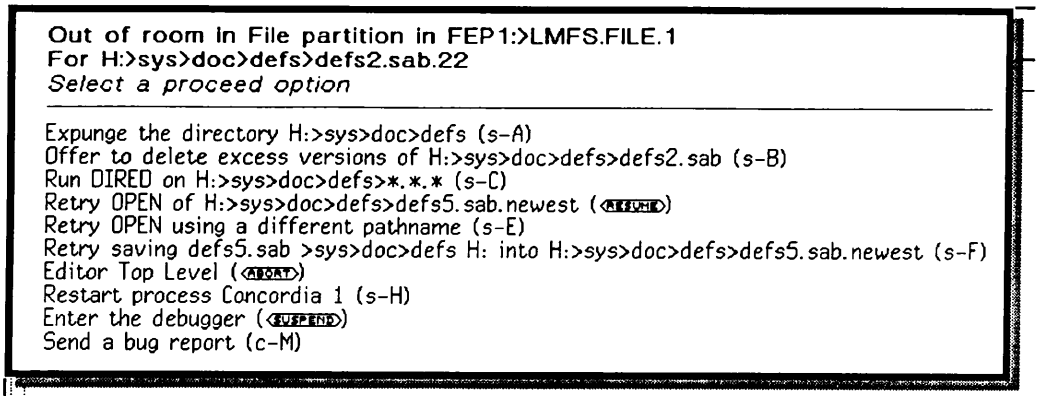


Figure 5. The Debugger Menu

7.1.4. Changing the Character Style of the Bug Banner is Now Possible

There is a new variable that controls the character style of the system information inserted at the beginning of a bug message.

dbg:*character-style-for-bug-mail-prologue* *Variable*

Creates the bug-report banner inserted into the text of bug messages, enabling you to choose the font. The default is nil.nil.tiny, specifying a small font for the bug-report banner.

To display a bug-report banner in a small font you can type the following:

```
(setq dbg:*character-style-for-bug-mail-prologue*
      (si:character-style-for-device-font 'fonts:quantum si:*b&w-screen*))
```

To display a bug-report banner in a large font you can type the following:

```
(setq dbg:*character-style-for-bug-mail-prologue*
      (si:parse-character-style '(nil nil :huge)))
```

You can also type the following to specify a particular font:

```
(setq dbg:*character-style-for-bug-mail-prologue* '(nil nil :huge))
```

This affects all the commands that send bug mail:

- c-M in the debugger.
- Report Bug Command Processor command.
- Bug (m-X) Zmail and Zmacs commands.
- **(zl:bug)**

7.2. Incompatible Changes to the Debugger in Genera 7.2

7.2.1. dgb:fun Code Change

In previous releases, the documentation for the function **dbg:fun** stated that it returned the function object of the current stack frame. In actuality, **dbg:fun** returned the function name.

In Genera 7.2, the code is changed so that **dbg:fun** returns the function object.

8. Changes to the User Interface in Genera 7.2

8.1. New Features in the User Interface in Genera 7.2

tv:key-test Function Again Available

The function **tv:key-test** is implemented in Release 7.2. It was available in Release 7.0, but not in Release 7.1.

8.1.1. Free Standing Mail Buffers Are Now Retrievable

When you use one of the non-Zmail, non-Zmacs bug-reporting mail commands (Report Bug in the CP or c-M in the Debugger), a Zmacs buffer is created called ***Bug-Mail-Frame-Mail-*n****. This contains the draft of the message so you can return to it later if you need to add information or just retransmit it.

8.1.2. You Can Now Customize Your SELECT Key Assignments

There is a new facility that allows you to change SELECT key assignments or to add new selectable activities to the SELECT key. See the section "Customizing the SELECT Key" in *Genera User's Guide.*

8.1.3. Miscellaneous New User Interface Features

- The interface of the Copy World command has been improved. See the section "Copy World Command" in *Genera Handbook.*
- c-X c-J now has a history.
- The Terminal VT100 simulator has been improved.
- There are many enhancements to Carry Tape and three new CP commands to use it:

  See the section "Read Carry Tape Command" in *Genera Handbook.*
  See the section "Write Carry Tape Command" in *Genera Handbook.*
  See the section "Show Carry Tape Command" in *Genera Handbook.*

- Font loading is improved.
- Variable line height management for windows is now more like the editor's.
- Users can now customize windows that are precreated.

- Custom keyboard mapping tables are now supported.
- Clear History works better.
- There is a simple mechanism for multiplexing of the mode lock key.
- Terminal simulator defaulting is improved.
- LGP2 font memory management is improved.

8.1.4. New Facilities in Genera 7.2

The following new user-interface facilities have been added:

- **dw:connect-graph-nodes**
- **dw:command-error**
- **dw:remove-window-typeout-window**
- **cp:*default-command-accelerator-echo***
- **dw:accept-values-fixed-line**
- **dw:accept-values-command-button**
- **dw:define-subcommand-menu-handler**

The following improvements have been made to existing facilities:

- The keyword option **:save-cursor-position** has been added to **dw:do-redisplay**.
- The keyword option **:selectable** has been added to **dw:define-program-framework**.
- The keyword option **:tester** has been added to **dw:define-command-menu-handler**.
- The keyword option **:menu-documentation** has been added to **dw:define-program-command**.
- The following options have been added to program frameworks: **:inherit-from**, **:window-wakeup**, and **:sub-presentation**.
- Incremental redisplay of graphs has been improved.
- **:typeout-window** and **:flavor** are now allowed together.

- Control over what goes into the history and what goes on the screen has been generalized.
- **with-underlining** has been made more flexible.
- Graph connection drawing has been made more flexible.
- **:clear-window** inside **dw:with-own-coordinates** works as it would work in a static window.
- A sample mechanism has been added to **dw:accept-values-sample**.
- **:freshline** inside incremental redisplay is disallowed.
- The presentation type description for large enumeration types has been improved.
- The programmer now has more control over accept-values caching.
- Keyboard commands are now allowed in **:accept-values** panes.
- Command menu centering has been made more flexible.

8.1.5. Improvements to User Interface Documentation

The following facility definitions have been added:

dw:remove-window-typeout-window *window* &key *(:prompt-p* **t***)* *Function*

Removes the typeout window from *window*. The removal is immediate if the keyword option **:prompt-p** is **nil**; if **:prompt-p** is **t**, the message "Type any character to refresh this display: " is displayed, and the window is removed when the user presses a character.

dw:after-program-frame-activation-handler *frame*

This is a generic function created for your program by **dw:define-program-framework**. You can define for it a method that will be called whenever your program frame is activated. The context in which your method will be called is unpredictable, so you should define it only to do simple things like set flags. For more information on this and like functions, See the section "Handling Asynchronous Window System Events" in *Programming the User Interface -- Concepts*.

dw:after-program-frame-selection-handler *program frame* *Function*

This is a generic function created for your program by **dw:define-program-framework**. You can define for it a method that will be called whenever your program frame is selected. The context in which your

method will be called is unpredictable, so you should define it only to do simple things like set flags. For more information on this and like functions, See the section "Handling Asynchronous Window System Events" in *Programming the User Interface -- Concepts.*

dw:before-program-frame-deactivation-handler
frame
This is a generic function created for your program by **dw:define-program-framework**. You can define for it a method that will be called whenever your program frame is deactivated. The context in which your method will be called is unpredictable, so you should define it only to do simple things like set flags. For more information on this and like functions, See the section "Handling Asynchronous Window System Events" in *Programming the User Interface -- Concepts.*

dw:before-program-frame-deexpose-handler
frame
This is a generic function created for your program by **dw:define-program-framework**. You can define for it a method that will be called whenever your program frame is deexposed. The context in which your method will be called is unpredictable, so you should define it only to do simple things like set flags. For more information on this and like functions, See the section "Handling Asynchronous Window System Events" in *Programming the User Interface -- Concepts.*

dw:before-program-frame-kill-handler *program* *Function*
frame
This is a generic function created for your program by **dw:define-program-framework**. You can define for it a method that will be called whenever your program frame is killed. The context in which your method will be called is unpredictable, so you should define it only to do simple things like set flags. For more information on this and like functions, See the section "Handling Asynchronous Window System Events" in *Programming the User Interface -- Concepts.*

dw:set-program-frame-configuration
&optional *(frame* **dw:*program-frame***)
Sets the configuration of program frame *frame* to the configuration specified by *configuration-name.*

8.2. Incompatible Changes to the User Interface in Genera 7.2

:data-arguments-are-disjoint Option to define-presentation-type Has New Default

The default value for the **:data-arguments-are-disjoint** option to **define-presentation-type** has been changed from **t** to **nil**.

Using **:data-arguments-are-disjoint nil** for a previously defined presentation type does not cause any code to cease working. If the data arguments really are disjoint, the code that searches for handlers will simply not take advantage of the fact and will consequently search more handlers than necessary. In those cases where this is the case and where it is important to optimize the search, you can redefine the presentation type to set **:data-arguments-are-disjoint** to **t**.

Pane State of :accept-values Pane

The state of the **:accept-values** pane in a program framework has been moved from the pane itself into the program.

Old and New Font Editors Can Coexist

Genera includes a new font editor, which replaces the older one. The older font editor, which is not included in the system for Release 7.2, can be loaded separately and can coexist with the new editor. The two editors have the same basic functionality.

GC Enhancements

Some GC enhancements to help with larger data structures have been made. The internal data structure returned by `(cdr (sys:cca-extra-info (sys:compiled-function-cca <compiled-function>)))` is no longer in the same format as the one **debugging-info** returns. Instead it is compressed. **debugging-info** is now allowed to cons. If you have a something with a **si:debug-info** property that is your own declaration, it is probably better to use the new function **si:debugging-info-user-entry** than **user::assq** on **debugging-info**.

Dumping of who-calls macros-expanded Allowed

The dumping of **who-calls macros-expanded** is now allowed for the distribution world. The **macros-expanded** properties used by **who-calls** are not in the world by default. **si:enable-who-calls**, when invoked, reloads a special file first which contains this information.

Simple Defstruct Accessor Functions Now All the Same

All simple defstruct accessor functions are now the same. This saves a few hundred pages in the world by eliminating duplicate inline compiled functions. Nothing is really lost in debuggability, since the functions are inline anyway. `c-sh-A` and `m-.` still work fine. If you look at the function stored in a structure accessor, you will find things with names like structure-array-leader-7.

New Function dw:set-program-frame-configuration

The function **dw:set-program-frame-configuration** adds a standard mechanism for changing configurations in a program frame.

Supdup to Waits

SUPDUP to WAITS is now able to use the meta key correctly. This means that SUPDUP between a Lisp machine running Release 7.1 and one running Release 7.2 will not work properly. Use 3600-LOGIN instead.

8.3. User Interface Bugs Fixed in Release 7.2

- Mouse handlers defined with **define-presentation-action** are correctly saved when a world is saved so that they are available again when the world is booted.
- Filling with m-Q is much faster.
- c-Left for marking in dynamic windows is easier to use.
- The printer argument to the :Output Destination keyword for CP commands now returns an instance so more things work.
- There is now a **:character-style** option for labels.
- Formatted output variable snapshotting now works properly.
- Several mouse handler gestures now work consistently.
- Several problems with source-level debugging have been eliminated.
- Flavor and generic function name completion has been improved.

8.4. The Genera 7.2 Graphics Substrate

Genera 7.2 introduces a more complete extension of the new graphics output substrate introduced in 7.0. Some of the major features include:

- It is upwardly compatible with the previous version.
- It is generic.
- It has a complete imaging model.

8.4.1. Compatibility of New Graphics Substrate

The new graphics substrate is almost completely compatible with previous versions.

- All messages to windows remain completely unchanged. In any instance where the new substrate required slightly different behavior, a new method implements the change while the old method remains undisturbed.

- All functions in the **graphics:** package are upwardly compatible with those introduced in Genera 7.0, with one exception. There are many new options, and quite a number of cases that did not work now behave consistently. The one function whose calling sequence has been changed is **graphics:draw-ellipse**. For an explanation of this: See the section "Incompatible Calling Sequence of **graphics:draw-ellipse**", page 132.

- The exact pixels affected by **graphics:draw-** functions are not always the same as before, because the scan conversion methods have been tightened up. For a more detailed explanation of this: See the section "Details of Scan Conversion", page 128.

8.4.2. Definition of a Generic Graphics Substrate

The new graphics substrate is *generic*. This means that there is a uniform interface for accessing the different facilities provided by different graphical output devices.

The graphical output devices supported in Genera 7.2 are:

- The standard bitmap screen and its color analogues

- The LGP2 laser printer

- Raster arrays intended for copying to the screen at some later time.

To say that the substrate is generic does not mean that all programs written for one device using the new substrate will run on all devices. In particular, there is no guarantee that every function will degenerate cleanly when the functionality requested of it does not exist. Every attempt has been made to do this where it is practical and reasonable. For instance:

- Using the **:color** option on a black and white screen produces a stipple pattern approximating the intensity of the desired color.

- Drawing with the **:opaque nil** option on the LGP2 draws special images to simulate this, even though the PostScript imaging model does not normally support color mixing.

In other cases, the limitations of the device are just too great.

- The LGP2 is not capable of drawing in **:alu :flip**, because it does not necessarily retain a complete image of the current partially drawn page.

- Graphics sent to character-only devices, for example, **with-output-to-string** or ASCII terminals, is not simulated by a clever use of *, +, |, -, / and \.

If you have any concerns about the compatibility of the drawing options you are using with the devices you need to output onto, you should check the corresponding dictionary entries.

8.4.3. The New Graphics Imaging Model

The new graphics output facilities provide a complete imaging model. All output can be done under a general transform that allows for scaling, rotating, and translating. Shapes can be drawn filled or unfilled, with or without an outline of arbitrary thickness. There is a uniform means of specifying texture attributes of the shape drawn, such as colors or stipple patterns.

In general, the mathematical shape of an object is speficied by the function used and its positional arguments. The details of how this shape is imaged are specified by keyword arguments.

The new imaging model is compatible with modern graphics imaging standards, in particular PostScript[TM] and the X Window System[TM].

8.4.4. Graphics Output to Windows

Genera 7.2 provides a number of enhancements to the graphics methods on windows to allow for implementation of the new imaging model. Particularly noteworthy are:

- The output history of graphics into a Dynamic Window now preserves temporal ordering of primitive operations. This means that if you draw several things on top of one another, the drawing will look the same when you scroll it off screen and back on again.

- There is now an exact definition for scan conversion of shapes in windows.

- Not all shapes will look the same as they did in 7.0. In particular, small circles have a more octagonal appearance by default. This is only true of the **graphics:draw-** functions, not of any messages. In all cases, it is possible to draw exactly what was output in 7.0, if you so desire, without reverting to use of the **:draw-** messages. For more details: See the section "Details of Scan Conversion", page 128.

8.4.4.1. Introduction to Scan Conversion in Windows

The screen is a *raster* device. This means that it is drawn on by means of an array of single units called *pixels*, each of which represents the contents of a single cell of bitmap memory. The most important thing to keep in mind when trying to

understand scan conversion is that these pixels are actually little squares that have finite extent: they are a single device unit on a side. Mathematical shapes are made up of points, which have no size. The business of scan conversion is to take the infinitely thin outline of a mathematical shape and determine which pixels should be affected in order to draw it.

Generally, a pixel is affected by drawing a shape when it is inside that shape. Recalling that pixels are little squares of finite dimension, you can see that most of the time there will be a number of pixels that are only partially within the shape to be drawn. It is important that there by an exact decision procedure to determine which of these pixels to draw.

The definition used by windows is as follows:

> A pixel is inside a shape, and hence affected when drawing that shape, if the center of the pixel is inside the shape. If the center of the pixel lies exactly on the boundary of the shape, it is considered inside if the inside of the shape is immediately to the right of the center point (increasing x direction). If the center of the pixel lies exactly on a horizontal boundary, it is considered inside if the inside of the shape is immediately below (increasing y direction).
>
> An unfilled shape is drawn by generating new outlines consisting of those points which are within 1/2 the thickness (normal distance) from the outline curve of the corresponding filled shape, and filling in the new outlines by applying the definition above.

This definition is compatible with that used by the X Window System. Even though X windows does not deal with fractional coordinates and the Genera graphics substrate does, there is no problem in generalizing the definition to accomodate that case. It is important to note that the decision point used for insideness checking is offset from the point used for addressing the pixel by half a device unit in both the x and y directions.

It is worth mentioning that for pixel devices that can store or display a single pixel cell in more than one color or graytone, one usually attempts to draw a given pixel proportionately lighter or darker depending on the fraction of the pixel within the shape. This is known as *anti-aliasing*. The new 7.2 graphics output substrate for windows does not support anti-aliasing, because the black and white screen is still the normal output device for Genera applications.

For more details on scan conversion: See the section "Details of Scan Conversion", page 128.

8.4.4.2. Summary of Differences in Graphics Output Functions

There are a number of differences between the Genera 7.2 **:draw-** messages and earlier versions of those messages:

- An argument that is an angle is interpreted the same way that a value returned by the **atan** function is interpreted. Since in a window increasing *y* is down the page, this means that angles with positive sines also point down the page. This is different from the way angles are drawn in an ordinary Cartesian coordinate system. It is also different from the way the **:draw-circular-arc** message interprets angles. The best way out of this confusion is to use a local coordinate system that points the way you expect, such as with **graphics:with-room-for-graphics**.

- Circles are not always the same shape as they were before. For more details on this: See the section "Details of Scan Conversion", page 128.

- Some arguments to **:draw-simple-ellipse** cause it to create figures with gaps. **graphics:draw-ellipse** does not have this problem.

- **:draw-line** does not always clip properly. For more details: See the section "Clipping of Thin Lines", page 133. The new functions involve more overhead than using the low-level messages directly. Once other factors, such as maintenance of a Dynamic window history are factored in, the difference amounts to about five to ten percent. For more details: See the section "Performance of Graphics Output", page 120.

8.4.5. Performance of Graphics Output

There are many considerations affecting the performance of a graphics output program. Ways of drawing that do not differ substantially in their specification (and hence in what they look like), can differ significantly in how they perform.

Some of the factors affecting speed of output are:

- The modularity level at which your program interfaces.

  ° The **:draw-rectangle** method on a window generates an error at a rather embarrassing time when passed floating point or ratio coordinates. The **graphics:draw-rectangle** function has no such restriction. Even when passed integer coordinates, this function has to verify that they are integers. When drawing lots of rectangles, this moderately small amount of time can still be noticeable.

  The algorithm used for drawing.

  ° A **:thickness** parameter of 0 is interpreted by the graphics substrate to mean not a line that is too thin to ever be seen, but the thinnest line which does indeed display visibly. This interpretation is compatible with PostScript and the X Window System. Additionally, as with X windows, a thin line (one of thickness 0) is allowed to be drawn using a faster algorithm which does not obey the complete requirements for raster scan conversion. A line of thickness 1 device unit, which is approximately as thick, must be scan converted exactly, which entails a much slower algorithm.

For more details on this: See the section "Details of Scan Conversion", page 128.

- A circle whose center and radius are integers or integers plus 1/2 can be imaged without using any fractional arithmetic. Circles that have fractional parameters need a slower, more general algorithm.

- How the algorithm is implemented.
  - Because of compatibility considerations, it was not possible to change any of the microcoded drawing routines or to add new ones. As a result, some drawing methods are in macrocode and other have greater microcode assistance.

The following two tables present performance of typical graphics operations. The units of the table entries are execution times in microseconds and, in pararentheses, the ratio of execution time relative to nominal time, that is, to the amount of time taken for the fastest equivalent operation in a static window.

8.4.5.1. Table of Relative Performance of Messages and Functions

Method	*Static*	*Dynamic*	*Constant Absolute Coordinates* [4]	*Encapsulated for Cartesian Block Output* [5]
(:draw-point 100 100)				
	250	2491 (10.0)	2193 (8.8)	n/a
(graphics:draw-point 100 100)				
	350 (1.4)	2250 (9.0)	4292 (17)	2327 (9.3)
(:draw-line 10 10 110 10)				
	385	2591 (6.7)	2726 (7.1)	n/a
(graphics:draw-line 10 10 110 10)				
	419 (1.1)	3032 (7.9)	4952 (13)	2893 (7.5)
(:draw-line 10 10 10 110)				
	1081 (2.8)	3207 (8.3)	3660 (9.5)	n/a
(graphics:draw-line 10 10 10 110)				
	1132 (2.9)	3441 (8.9)	5710 (15)	3994 (10)
(:draw-line 10 10 110 110)				
	1196 (3.1)	3717 (9.7)	3913 (10)	n/a
(graphics:draw-line 10 10 110 110)				
	1264 (3.3)	3574 (9.3)	5794 (15)	3741 (9.7)
(:draw-line -3 -10 25 57)				
	843 (2.2)	3031 (7.9)	3186 (8.3)	n/a
(graphics:draw-line -3 -10 25 57) [3]				
	15232 (40)	17495 (45)	20084 (52)	18253 (47)
(:draw-dashed-line 10 10 110 10)				

2741	5257 (1.9)	5387 (2.0)	n/a
(graphics:draw-line 10 10 110 10 :dashed t)			
2847 (1.0)	5336 (1.9)	5681 (2.1)	5030 (1.8)
(:draw-dashed-line 10 10 10 110)			
3024 (1.1)	5599 (2.0)	6061 (2.2)	n/a
(graphics:draw-line 10 10 10 110 :dashed t)			
3467 (1.3)	5689 (2.1)	6253 (2.3)	5664 (2.1)
(:draw-dashed-line 10 10 110 110)			
4014 (1.5)	6833 (2.5)	7133 (2.6)	n/a
(graphics:draw-line 10 10 110 110 :dashed t)			
8976 (3.3)	29937 (11)	30521 (11)	30319 (11)
(:draw-lines :draw 100 100 100 200 200 200 200 100 100 100)			
2925	12256 (4.2)	12874 (4.4)	n/a
(graphics:draw-lines (100 100 100 200 200 200 200 100 100 100))			
5060 (1.7)	14677 (5.0)	15335 (5.2)	15372 (5.3)
(:draw-lines :draw 200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)			
7618	38163 (5.0)	42464 (5.6)	n/a
(:draw-curve #(200 100 100 200 300 400 400 300 200) #(100 200 300 400 400 300 200 100 100))			
7747 (1.0)	40830 (5.4)	32606 (4.3)	n/a
(:draw-closed-curve #(200 100 100 200 300 400 400 300) #(100 200 300 400 400 300 200 100))			
7819 (1.0)	32342 (4.2)	30870 (4.1)	n/a
(graphics:draw-lines (200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100))			
10374 (1.4)	34969 (4.6)	35434 (4.7)	35266 (4.6)
(graphics:draw-lines (200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100) :closed t)			
10567 (1.4)	41664 (5.5)	35962 (4.7)	36219 (4.8)
(:draw-rectangle 100 100 10 10)			
1476	11461 (7.8)	3518 (2.4)	n/a
(graphics:draw-rectangle 10 10 110 110) [2]			
1574 (1.1)	3527 (2.4)	5900 (4.0)	5700 (3.9)
(:draw-triangle 0 100 200 100 100 20)			
1747	4018 (2.3)	6301 (3.6)	n/a
(graphics:draw-triangle 0 100 200 100 100 20) [2]			
1847 (1.1)	4166 (2.4)	6278 (3.6)	5691 (3.3)
(:draw-circle 100 100 20)			
8566	10487 (1.2)	10578 (1.2)	n/a
(graphics:draw-circle 100 100 20 :filled nil)			
15546 (1.8)	18668 (2.2)	19615 (2.3)	18582 (2.2)

(:draw-circular-arc 100 100 20 0.0 1.0)
4188 13678 (3.3) 29057 (6.9) n/a
(graphics:draw-circle 100 100 20 :filled nil :end-angle 1.0)
13591 (3.2) 21718 (5.2) 22635 (5.4) 21592 (5.2)

(:draw-filled-in-circle 100 100 20)
4489 8837 (2.0) 6511 (1.5) n/a
(graphics:draw-circle 100 100 20)
9991 (2.2) 13289 (3.0) 13979 (3.1) 12934 (2.9)
(graphics:draw-circle 100.5 100.5 20.5)
12249 (2.7) 15383 (3.4) 15982 (3.6) 15389 (3.4)

(:draw-filled-in-sector 100 100 20 0.0 1.0)
14725 23444 (1.6) 23603 (1.6) n/a
(graphics:draw-circle 100 100 20 :end-angle 1.0)
17654 (1.2) 26353 (1.8) 27415 (1.9) 27148 (1.8)

(:draw-ring 100 100 20 40)
15989 18058 (1.1) 18175 (1.1) n/a
(graphics:draw-circle 100 100 40 :inner-radius 20)
24968 (1.6) 27950 (1.7) 28575 (1.8) 28004 (1.8)

(:draw-simple-ellipse 100 100 40 20)
14067 16122 (1.1) 16211 (1.2) n/a
(graphics:draw-ellipse 100 100 40 20 :filled nil)
21481 (1.5) 24597 (1.7) 25252 (1.8) 24539 (1.7)

(:draw-filled-in-simple-ellipse 100 100 40 20)
7147 11082 (1.6) 9246 (1.3) n/a
(graphics:draw-ellipse 100 100 40 20)
9956 (1.4) 12976 (1.8) 13615 (1.9) 13348 (1.9)

(:draw-polygon :draw 200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)
52952 70297 (1.3) 73994 (1.4) n/a
(graphics:draw-polygon (200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100))
55593 (1.0) 74770 (1.4) 73149 (1.4) 73808 (1.4)
(:draw-convex-polygon :draw 200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)
23338 (0.44) 44198 (0.83) 47628 (0.90) n/a
(graphics:draw-polygon (200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100) :points-are-convex-p t)
26547 (0.50) 49192 (0.93) 49595 (0.94) 50442 (0.95)

(:draw-regular-polygon 100 100 200 100 5)

	8685	17743 (2.0)	23938 (2.8)	n/a
(graphics:draw-regular-polygon 100 100 200 100 5)				
	21319 (2.5)	29393 (3.4)	34081 (3.9)	31075 (3.6)
(:draw-wide-curve #(100 100 200 200 100) #(100 200 200 100 100) 10)				
	13798	17278 (1.3)	20647 (1.5)	n/a
(graphics:draw-lines (100 100 100 200 200 200 200 100 100 100) :thickness 10) [1]				
	8663 (0.63)	24919 (1.8)	31779 (2.3)	28693 (2.1)
(:draw-wide-curve #(200 100 100 200 300 400 400 300 200) #(100 200 300 400 400 300 200 100 100) 10)				
	32427	36006 (1.1)	38633 (1.2)	n/a
(graphics:draw-lines (200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100) :thickness 10)				
	292454 (9.0)	393573 (12)	397503 (12)	392842 (12)
(:draw-cubic-spline #(200 100 100 200 300 400 400 300) #(100 200 300 400 400 300 200 100) 20)				
	404910	1543958 (3.8)	1612355 (4.0)	n/a
(graphics:draw-cubic-spline (200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100))				
	434411 (1.1)	1590461 (3.9)	1717139 (4.2)	1656092 (4.1)

Notes:

1. The new substrate is faster because its higher level of expression lends itself better to optimization.

2. The small amount of extra time is due to checking for non-integer coordinates, to avoid getting an error that is hard to recover from.

3. This is not microcode assisted well.

4. In these cases, the method is wrapped in a **dw:with-own-coordinates** form. This uses an encapsulating stream that has a large setup time, so the speed factors are not really representative of many operations to the same stream.

5. In these cases, the method is wrapped in a **graphics:with-room-for-graphics** form.

8.4.5.2. Table of Relative Performance of Drawing Modes

Function	*Exact*	*Integer*	*Center*
(graphics:draw-line 10 10 110 10)			
	419	419	419

(graphics:draw-line 10 10 110 10 :thickness 0)			
	420 (1.0)	419	464 (1.1)
(graphics:draw-line 10 10 110 10 :thickness 1)			
	2749 (6.6)	3123 (7.5)	3123 (7.5)
(graphics:draw-line 10 10 110 10 :thickness 2)			
	2592 (6.2)	2964 (7.1)	2965 (7.1)
(graphics:draw-line 10 10 110 10 :thickness 10)			
	2727 (6.5)	3099 (7.4)	3099 (7.4)
(graphics:draw-line 10 10 10 110)			
	1192	1132 (0.95)	1132 (0.95)
(graphics:draw-line 10 10 10 110 :thickness 0)			
	1176 (0.99)	1176 (0.99)	1132 (0.95)
(graphics:draw-line 10 10 10 110 :thickness 1)			
	3530 (3.0)	3929 (3.3)	3884 (3.3)
(graphics:draw-line 10 10 10 110 :thickness 2)			
	3380 (2.8)	3753 (3.1)	3753 (3.1)
(graphics:draw-line 10 10 10 110 :thickness 10)			
	3380 (2.8)	3753 (3.1)	3799 (3.2)
(graphics:draw-line 10 10 110 110)			
	1309	1309	1308 (1.00)
(graphics:draw-line 10 10 110 110 :thickness 0)			
	1308 (1.00)	1264 (0.97)	1264 (0.97)
(graphics:draw-line 10 10 110 110 :thickness 1)			
	36368 (28)	4163 (3.2)	4182 (3.2)
(graphics:draw-line 10 10 110 110 :thickness 2)			
	50532 (39)	7094 (5.4)	7145 (5.5)
(graphics:draw-line 10 10 110 110 :thickness 10)			
	60637 (46)	7328 (5.6)	7327 (5.6)
(graphics:draw-rectangle 210 210 310 310)			
	1576	1576	1576
(graphics:draw-rectangle 0 0 100 100 :translation (200 200) :stretch-x 1 :rotation 0)			
	3746 (2.4)	3747 (2.4)	3791 (2.4)
(graphics:draw-rectangle 0 0 100 100 :translation (200 200) :stretch-x 1 :rotation 1)			
	75980 (48)	8655 (5.5)	8654 (5.5)
(graphics:draw-rectangle 0 0 100 100 :translation (200 200) :stretch-x 2 :rotation 0)			
	4051 (2.6)	4051 (2.6)	4051 (2.6)
(graphics:draw-rectangle 0 0 100 100 :translation (200 200) :stretch-x 2 :rotation 1)			
	116784 (74)	11290 (7.2)	11319 (7.2)
(graphics:draw-triangle 0 100 200 100 100 20)			
	1848	1864 (1.0)	1847 (1.00)

(graphics:draw-regular-polygon 100 100 200 100 5)
18070 17994 (1.00) 17950 (0.99)

(graphics:draw-circle 200 200 20)
10115 10087 (1.00) 12342 (1.2)
(graphics:draw-circle 0 0 20 :translation (200 200) :stretch-x 1 :rotation 0)
11119 (1.1) 11145 (1.1) 13393 (1.3)
(graphics:draw-circle 0 0 20 :translation (200 200) :stretch-x 1 :rotation 1)
13081 (1.3) 13078 (1.3) 15324 (1.5)
(graphics:draw-circle 0 0 20 :translation (200 200) :stretch-x 2 :rotation 0)
11741 (1.2) 11752 (1.2) 17537 (1.7)
(graphics:draw-circle 0 0 20 :translation (200 200) :stretch-x 2 :rotation 1)
55330 (5.5) 54726 (5.4) 60191 (6.0)

(graphics:draw-circle 200 200 20 :filled nil)
15560 15556 (1.00) 15862 (1.0)
(graphics:draw-circle 0 0 20 :filled nil :translation (200 200) :stretch-x 1 :rotation 0)
16810 (1.1) 16761 (1.1) 18773 (1.2)
(graphics:draw-circle 0 0 20 :filled nil :translation (200 200) :stretch-x 1 :rotation 1)
18586 (1.2) 18772 (1.2) 18840 (1.2)
(graphics:draw-circle 0 0 20 :filled nil :translation (200 200) :stretch-x 2 :rotation 0)
23224 (1.5) 23361 (1.5) 23422 (1.5)
(graphics:draw-circle 0 0 20 :filled nil :translation (200 200) :stretch-x 2 :rotation 1)
216606 (14) 210840 (14) 216866 (14)

(graphics:draw-cubic-spline
(200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)
:start-relaxation :cyclic :thickness 0)
556372 568649 (1.0) 676032 (1.2)
(graphics:draw-cubic-spline
(200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)
:start-relaxation :cyclic :thickness 1)
3625544 (6.5) 1016596 (1.8) 1042373 (1.9)
(graphics:draw-cubic-spline
(200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)
:start-relaxation :cyclic :thickness 2)
3316662 (6.0) 1203678 (2.2) 1294307 (2.3)
(graphics:draw-cubic-spline
(200 100 100 200 100 300 200 400 300 400 400 300 400 200 300 100 200 100)

:start-relaxation :cyclic :thickness 10)
3280043 (5.9) 1152265 (2.1) 1156537 (2.1)

8.4.6. Choosing a Technique for Graphics Output

There are a number of things to be considered when choosing how to use the new graphics substrate.

- If you already have an application using the **:draw-** messages, it will continue to work. Especially if you have had to go to lots of trouble to determine the exact pixel locations and offsets of various figures, you may not have any motivation for changing it.

- If you want your drawing to work compatibly on the LGP2 and any future similar device which is supported, then you must use the new **graphics:draw-** facilities. You must also keep in mind the coordinate system issues.

  See the section "Graphics Output Coordinate Systems", page 128.

  Note that the LGP2 has high enough resolution that exact pixel choices are usually unimportant, and that Symbolics has no control over the speed of the Adobe code that runs in it, so the remaining decision points covered here apply to the screen only.

- If you are just doing a simple drawing, such as a pie chart, you probably do not need to think about any further refinement.

- If it is important that figures tile correctly, you should set the value of the keyword **:coordinate-mode** to **:exact**, or better yet do not use this keyword, since it defaults to **:exact**. Similarly, if you require that shapes with fractional coordinates be drawn properly rather than rounded to integer shapes, set **:coordinate-mode** to **:exact**.

- If you want small circles which appear symmetrical about a single pixel dot and do not need to carefully align them with other shapes, you can use **:coordinate-mode :center**.

- If you do need to align round circles with other shapes, you are probably best off adding in the 1/2-pixel offsets yourself so that you will add them exactly where needed.

- If speed is more important than tiling accuracy in sketching a figure, drawing with **:thickness 0** is the faster way.

- If speed is of moderate importance in drawing a filled figure or one with thick lines, and exactness is of little importance, then **:coordinate-mode :integer** will always use special integer drawing methods, which are faster. Keep in mind that it is better to do this by means of a single **graphics:with-coordinate-mode**

around all the output than to do it in each drawing function call, which requires more argument decoding.

- If speed is of paramount importance, you will save time in some cases by using the old **:draw-** messages. Of course, the application will have to take responsibility of conversion to integers, or errors will result, and no scaling or output to the LGP2 is available. As this is only a small gain over the fastest case of the **graphics:draw-** functions, it is also possible that even this technique will not meet your needs adequately.

8.4.7. Graphics Output Coordinate Systems

The initial coordinate system for the graphics functions is the device coordinate system. This system is not the same for all devices, so various facilities are provided for accessing it uniformly. The general graphics transformation facilities can be used to provide any scale that is convenient for a particular operation. Keep in mind that there is no restriction on the type of coordinate arguments. They can be integers, floats, or ratios.

The coordinate system of the LGP2 is a single page, with <0,0> at the bottom. The coordinate system of a dynamic window is a plane with <0,0> at the top of the history, and some greater *y* value at the top of the currently visible screenful, and some greater still *y* value at the current cursor position. It is very useful to be able to deal with graphics as a single block interspersed with any text output already on the page or in the output history. Additionally, it is usual to want to deal with graphics within a primary Cartesian quadrant, so that <0,0> is at the lower left. The special form provided for this is **graphics:with-room-for-graphics**. Note that the 0 point of the quadrant is at the bottom of the local coordinate system, not the bottom of the page or screenful.

The initial scale for the LGP2 is 1/72 inch, a "PostScript point." (The X Window System uses the printer's point, which is 0.01384 or about 1/72.27 inch.) There is more than one device pixel available within a single point. The initial scale of the screen is a single pixel. In order to draw a figure with a certain physical dimension, either inches/centimeters, or as a fraction of a page or number of lines on the device, the special form **graphics:with-physical-device-scale** is provided.

8.4.8. Details of Scan Conversion

The resolution limitations of a raster device make scan conversion a hard problem, for which there are no easy universal solutions. For an application in which exactness is important, it is critical that the same means be applied for all shapes without exception. The definition used by windows is borrowed from the X Window System, extended in the most obvious way to allow for non-integral coordinates. Even though this definition is standard, it is worth considering some of the motivations.

When two shapes share a common edge, it is important that only one of them own any pixel. Mathematically speaking, given two figures whose intersection is of measure zero, the intersections of their rasterizations should be of measure zero. Fig-

ure 6 illustrates this. The pixels along the diagonal belong to the lower figure. When the decision point within the pixel (the center) lies to one side of the line or the other there is no issue. When the boundary passes through the decision point, which side the inside of the figure is on is used to decide.

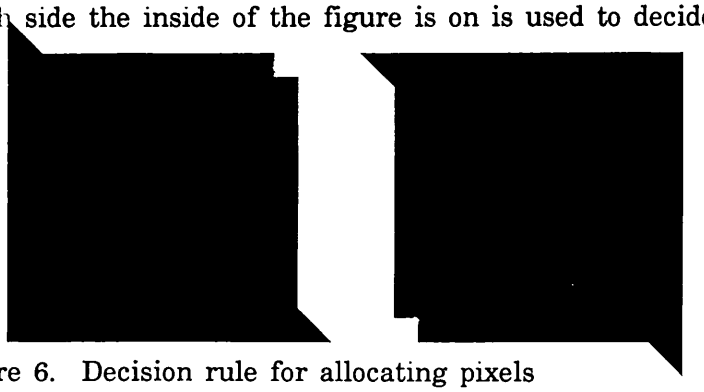
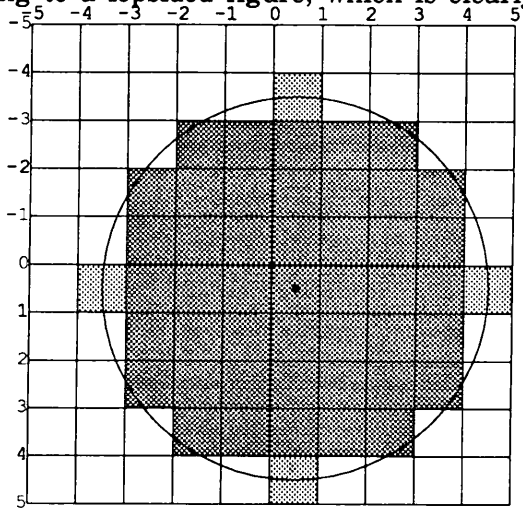


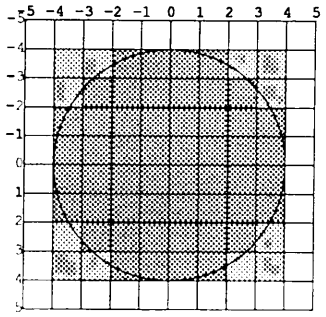
Figure 6. Decision rule for allocating pixels

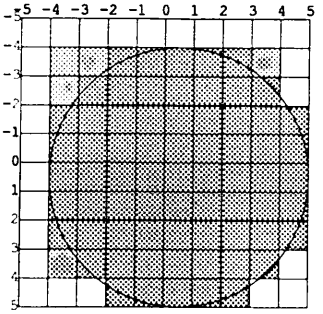
The reason for choosing the decision point half a pixel offset from the address point is to reduce the number of common figures that invoke the boundary condition rule. This leads to more symmetrical results. For instance, in the figure below, we see a circle drawn when the decision point is the same as the address point. The four lighter points are indeterminate: it is not clear whether they are inside or outside. Since we desire to have each boundary case determined according to which side has the figure on it, and since we must apply the same rule uniformly for all figures, we have no choice but to pick only two of the four points, leading to a lopsided figure, which is clearly undesirable.



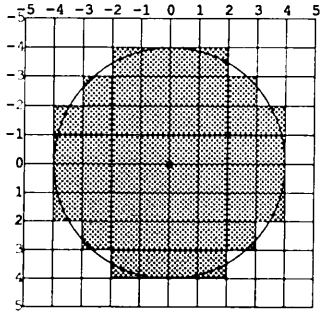
If we had instead chosen to take all four points, as the **:draw-filled-in-circle** method does, we would have a nice symmetrical figure. This figure is symmetrical about a whole pixel, however, so it is one pixel wider than it ought to be. The problem with this can be seen clearly if we attempt to draw a rectangle and circle overlaid.

```
(defun shape (r)
  (graphics:draw-circle 0 0 r)
  (graphics:draw-rectangle (- r) (- r) (+ r) (+ r) :alu :flip))
```



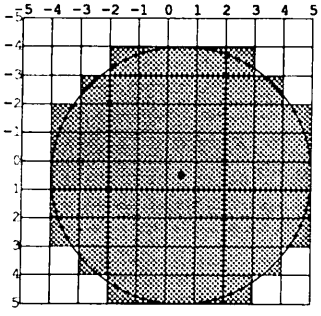


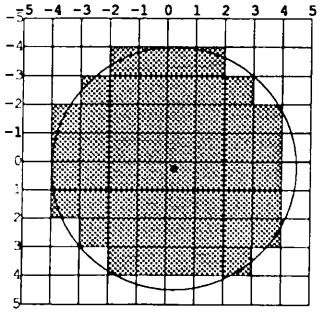
For this reason, we choose to have the decision point at the center of the circle. This draws circles that look like the figure below.



The problem with these circles is that they look rather more like stop-signs than the ones the old **:draw-circle** produced. This is due to the eye picking up on the straight lines before it picks up on the rotational symmetry. We sometimes prefer something that is indeed centered around a whole pixel. Once we understand the rules for scan conversion, it is easy to see how to draw this figure. We want a circle that is of radius *r*+1/2 (since we have one more pixel in the middle to account for) and whose center is at <1/2,1/2> (the point in the middle of the square about which we want to be symmetrical). Since the graphics functions are not restricted to integer coordinates, this is easy to do.

Since this is a rather common case, we provide a special drawing mode to enable this. This is **:coordinate-mode :center**. You can draw such circles either way.

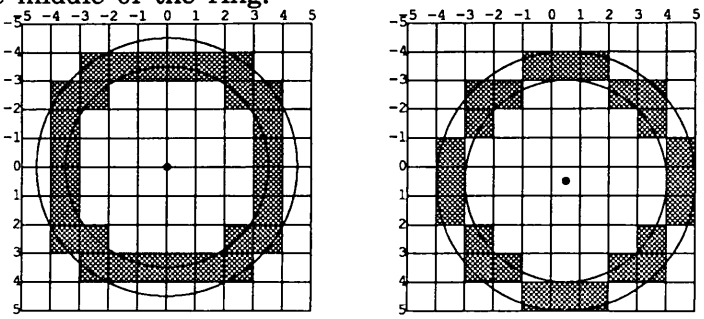




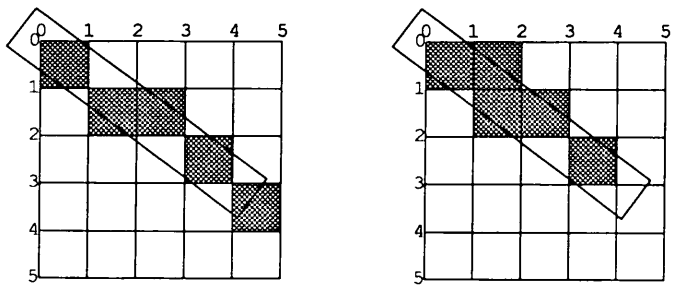
Despite the well-controlled symmetry of the above shapes, there is really no way to guarantee that all circles look round when scan converted. For instance, one whose coordinates are offset by one quarter unit does not look very round. For this reason, it is often desirable to make sure that coordinates and radii are integers if they come from some potentially inexact computation and you want the figure to look round.

Since this is also a rather common case, you can disable the exact conversion of fractional shapes with **:coordinate-mode :integer**.

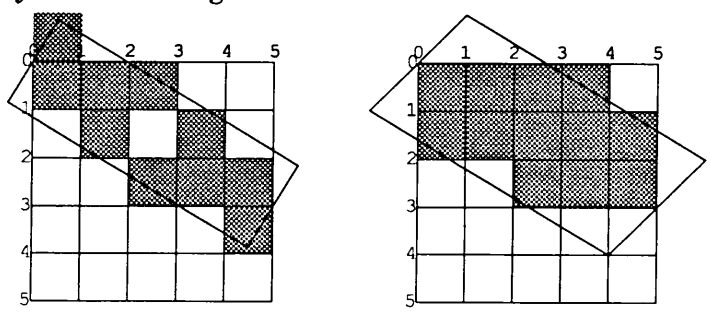
Unfilled circles follow exactly the same rules as filled circles. An unfilled circle is essentially a circular ring of the desired thickness with the circle in question lying in the middle of the ring.



We distinguish lines of thickness 0, which are drawn by a special fast integer slope algorithm, from those of thickness 1, which are exactly drawn as tilted rectangles. For the majority of cases, these produce results that are similar enough for most purposes.

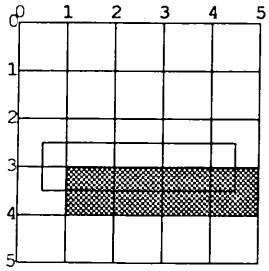


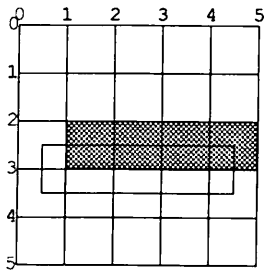
For thick lines, we can draw the exact tilted fractional rectangle, or we can round the coordinates of the rectangle so it becomes a polygon with integral coordinates and draw that. The latter case is faster since it can make full use of the triangle microcode. For the majority of cases, these two methods produce results that are usually similar enough.



The decision about which side of the figure to take when a boundary line passes through the decision point is made arbitrarily. We have chosen to be compatible with the X Window System definition, although this is not necessarily the most convenient. The main problem with this is illustrated by the case of a horizontal line. Our definition chooses to draw the rectangular slice above the coordinates, since those pixels are the ones whose centers have the figure immediately above them. It would be more convenient if we could recognize that a line from <1,3> to

<4,3> is the same as the primitive device rectangle of height 1 and width 4 at <1,3>.





We have attempted to choose a primitive imaging definition that is convenient for lower level control. We have then tried to add higher level facilities on top of it that capture some of the more common convenient interfaces. These modes are not the only possible modes that could be added. For instance, we could choose a model which did not attempt to tile correctly and always selected pixels that had any of the figure at all in them. Fortunately, the framework of the **graphics:** functions permits adding this modularly at a later time if the need can be shown and the details worked out.

We have chosen to make our definition compatible with the X Window System to permit future migration to a graphics system based on that system at the lowest level. Since X is designed for a healthy combination of generality and efficiency, it is hoped that this would lead to superior performance and generality, while remaining essentially compatible.

It would also be possible to define all operations in terms of a primitive path filling operation. This would take a shape as a set of continuous segments, such as lines, conic sections, and cubics. It would then scan convert the area within the segments, using standard techniques. This is essentially the primitive operation defined for the PostScript imaging system. This would require much less special-case code, but would not be as well optimized. Such a system could benefit, however, from the aid of a graphics coprocessor or special assistance microcode.

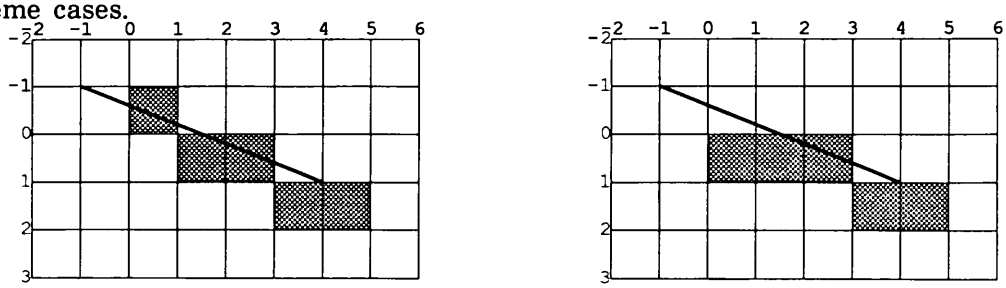
8.4.9. Incompatible Calling Sequence of graphics:draw-ellipse

The calling sequence of **graphics:draw-ellipse** has been changed from 7.0 to 7.2. The 7.0 specification uses the two axes as vectors. This interface makes it difficult for the ellipse drawer, because a general affine transform does not preserve axes of foci. An ellipse is transformed into another ellipse with the center at the transformed center, but that is about the limit to what one can say. For this reason, the specification has been changed to call for a rectilinear ellipse with only the axis lengths. In order to draw a tilted ellipse, you tilt the whole coordinate system, using the standard methods, including the **:rotation** keyword to the actual **graphics:draw-ellipse** function. This makes the job of scan converting much easier.

The new specification also allows defining the **graphics:draw-circle** function trivially in terms of **graphics:draw-ellipse**, and then applying optimization techniques back into circles *after transforming*, inside the latter function.

8.4.10. Clipping of Thin Lines

The **:draw-line** message does not clip properly. It always chooses a point on the edge of the window that is being clipped across and draws a line to that point, using the microcoded line drawer. However, the slope of this line need not be the same as the slope of the original line, so that the portion of the line within the window may be adversely affected. This is illustrated by the right-hand side of the figure, in which a point on the x=0 boundary has been chosen such that the slope is not that of the ideal line. For compatibility reasons, neither the **:draw-line** message nor the microcode drawer was changed. Instead, the **graphics:draw-line** function uses a different method that clips properly. However, it cannot always use the microcode line drawer to get all of the line, and may therefore be slower in some extreme cases.



8.4.11. Erasing versus Deleting versus Painting White

There are three different ways of getting rid of some graphics that you have drawn, each of which has its place. It is important to understand the difference and to know when a particular one is right for you.

1. Erasing, as with the **graphics:erase-rectangle** function, or the **:clear-region**, **:clear-rest-of-window**, or **:clear-history** messages to a window, means clearing a portion of the output history, removing anything that is in it. This is the fastest way to get rid of something that you have output. However, it is important to keep in mind that things are only removed from the history in single units. If you erase a region that intersects a graphical object, that object will be removed from the output history, even if it is not wholly contained in the region. For reasons of efficiency, only the area you requested to clear will be erased right away. Only when you scroll away from an area and back again will you notice that a partially erased graphic has been wholly removed from the history.

2. Deleting, as with **graphics:erase-graphics-presentation**, means removing something from the output history and redisplaying everything that overlaps it. Because proper temporary ordering of output is remembered as part of the history, when you delete something, it is possible to completely reconstruct what would appear on the screen as if you had never drawn it.

3. Painting in white means adding a new graphic to the output history on top of what is already there that obscures what is underneath it. If the shape in question does not completely cover something underneath, then the combined shape will display properly even when scrolled back.

8.4.12. Text as Graphics

There are several different interpretations of text as graphical output. The interesting issues arise when the graphics containing the text is scaled or rotated, and when the text is mixed with other graphics.

There are several possible actions that might be taken when a graphics transformation is applied to text:

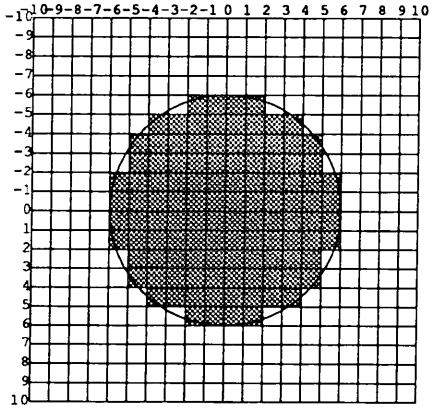
- Affect only the starting point of the text, keeping the actual letters upright.
- Affect the baseline of the text, slanting it, but keeping the letters upright.
- Actually scale and rotate the individual glyphs along with the rest of the drawing.

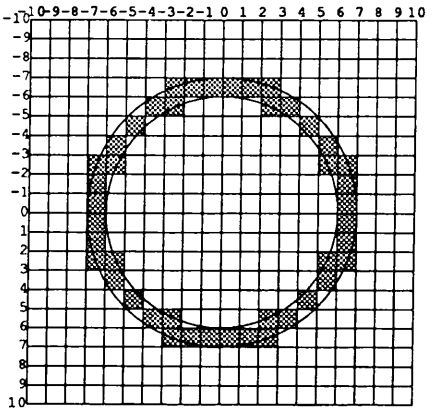
Since the various output devices available to the graphics substrate do not implement a consistent set of fonts, one must be careful when mixing text and graphics if the result is to be device- and scale-independent. There are two possibilities:

- Ask the stream the size of the text to be drawn and constrain the other graphics to that size.
- Use the kinds of text graphics drawing that allow scaling to a given size and specify that in accordance with the rest of the drawing.

8.4.13. Outlining with Thin Lines

Drawing a shape with **:thickness 0** is not the same as outlining it. For instance, take a circle of radius 6 and consider it and its unfilled counterpart. These two shapes overlap in a few pixels because of the overlap of the mathematical shapes. Keeping in mind the scan conversion model, it is not hard to generate an outline for this shape. You just draw an unfilled circle of radius 6.5 and thickness 1 with the same center. This shape is a ring with inner radius 6 and outer radius 7, which is outlines the filled circle.





Other simple shapes can be treated similarly. For more complicated shapes, there is no simple representation of the outline based on the filled or unfilled shape. A

wholly new mode just for outlining is required. This has not yet been implemented. There are, then, some shapes for which there is no convenient way to generate a closely surrounding and similarly shaped mouse-sensitivity box.

9. Changes to Zmail in Genera 7.2

9.1. Incompatible Changes to Zmail in Genera 7.2

- Mail files in KBIN format written in 7.2 cannot be loaded into earlier versions of Zmail. Mail files in KBIN format written in 7.0 or 7.1 can be read into 7.2 and are updated to the new format automatically when they are written out.
- The Next and Previous commands now skip deleted messages.
- The commands that prompt for comments now do so in a multi-line buffer, so you must terminate comments with END. RETURN no longer works to terminate a comment.

9.2. New Features in Zmail in Genera 7.2

9.2.1. New Commands Added to Mark Survey

Four new commands have been added to Mark Survey (under [Select] in the Zmail menu).

c-M	Marks all messages for inclusion in the collection. This is useful if you want to create a collection that includes all but a few messages in the current sequence.
c-sh-M	Unmarks all messages.
c-sh-N	Toggles the marked state of the current message and then moves the summary cursor forward to the next message. (Equivalent to Space followed by c-N but fewer keystrokes.)
c-sh-P	Toggles the marked state of the current message and then moves the summary cursor backward to the previous message.

See the section "Creating a Mail Collection by Marking Individual Messages" in *Communicating With Other Users.*

9.2.2. Zmail Now Knows About c-X

Zmail now has the appropriate presentation types to allow it to prompt for message sequences in a fashion compatible with prompting for buffer names in Zmacs. That is, you can just type the filename instead of the complete pathname. In addition, in most cases, when you are asked for a mail file buffer, you can also enter the name of any mail file mentioned in your Zmail profile that is not loaded yet.

Try pressing HELP when Zmail prompts you for sequences to see what is acceptable.

Thanks to the above change, Zmail now has c-X commands:

c-X B or [Select] (**zwei:com-zmail-select-sequence**)
Selects a message sequence. If you invoke it from the keyboard, it prompts for a sequence name in the minibuffer. Clicking Left selects the previously selected sequence. Clicking Middle creates a new collection by filtering. Clicking Right displays a menu of existing sequences, unloaded mail files, and special actions such as Read/Create file, Mark Survey, and so on.

c-X K (**zwei:com-zmail-kill-sequence**)
Kills a message sequence. It prompts for a sequence name in the minibuffer. If you ask to kill the current sequence, this command asks for the name of another sequence select as the current sequence. If you ask to kill a mail file buffer, you are asked either to save it first or to confirm that you picked the proper sequence.

c-X c-B (**zwei:com-zmail-list-sequences**)
Lists the current message sequences.

c-X c-F (**zwei:com-zmail-edit-mail-file**)
Edits a mail file. It prompts for the pathname of a mail file in the minibuffer and reads that mail file into a Zmail mail file buffer. If the mail file does not exist, an empty mail file buffer is created which, when saved, creates the mail file with the requested pathname.

c-X c-R (**zwei:com-zmail-examine-mail-file**)
Examines an existing mail file. It prompts for the pathname of a mail file in the minibuffer, reads that mail file into a Zmail mail file buffer, and then disables saving of the buffer. The mail file must already exist.

c-X c-S (**zwei:com-zmail-save-mail-file**)
Saves the current mail file. Without a numeric argument, it expunges deleted messages from the current mail file buffer and writes the updated buffer to the mail file. With a numeric argument, it starts saving the current mail file in a background process. Deleted messages are not expunged. The Abort Background Save command can be used to stop the background save.

c-X c-sh-F (**zwei:com-zmail-edit-file**)
Edits a file. Prompts in the mini-buffer for a pathname (the default is the first pathname if the File References Field, if any) and selects Zmacs to edit the file.

In addition to the above, c-m-L now invokes **zwei:com-zmail-select-previous-sequence**. When you give c-m-L a zero numeric argument, it prompts for a sequence name in the minibuffer after listing the possible sequences. The entries in the listing are, of course, mouse sensitive.

9.2.3. New Zmail Profile Options

zwei:*add-supersedes-and-comments-when-retransmitting* *Variable*

Profile Option: Add Supersedes and Comments fields when retransmitting a message.

Controls the addition of Supersedes: and Comments: fields to a message when you retransmit a draft after resuming it with the Continue (RESUME) command.

The choices are:

No	Never adds Supersedes: or Comments: fields.
Ask	The default. Asks for permission to add each field separately.
Supersedes only	Always adds a Supersedes: field but never adds a Comments: field.
Supersedes then ask	Always adds a Supersedes: field and then ask permission to add a Comments: field.
Yes	Always adds both fields.

See the section "Zmail Comments: Field".

The Comments: field is a multi-line field whose value is read from the minibuffer. The above profile option only controls whether or not Zmail will first ask for permission to read the Comments: in the minibuffer. If the Yes setting is chosen for the above option, Zmail will go directly to reading Comments: in the minibuffer without asking any questions.

zwei:*ask-before-executing-dangerous-zmail-commands* *Variable*

Profile Option: Ask permission before executing time consuming and/or destructive commands.

Controls whether Zmail asks for confirmation before starting a time consuming or potentially dangerous operation.

The choices are:

All	The default. All dangerous commands should ask for permission.
Selective	Only the dangerous commands listed in the "Commands which will ask permission" profile option ask for permission.

None	No dangerous commands ever ask for permission. This includes Map Over and means that the setting of "Maximum number of messages which may be mapped over without asking permission" is ignored. See the variable **zwei:*too-many-msgs-query-threshold***, page 141.

zwei:*autosave-if-inbox-requires-save* *Variable*

Profile Option: Automatically save the mail file if needed to read new mail.

Controls the whether or not your mail file is written out when you do Get Inbox on systems that do not support multiple renamed inboxes (UNIX and ITS).

The choices are:

Yes	Automatically write out the file when a Get Inbox is done.
No	Do not write out the mail file automatically. Get Inbox tells you that you cannot read new mail until you have saved your mail buffer.
Ask	Tells you that your mail file must be saved before getting new mail and asks your permission to save it.

zwei:*default-bcc-list* *Variable*

Profile Option: Default initial Bcc list

Allows you to specify a list of recipients of "blind carbon copies" of messages you send.

zwei:*draft-editor-end-key-treatment* *Variable*

Profile Option: Effect of END key in *Headers* window.

Controls what END does when you press it in the headers window when you are composing a message. The choices are:

Send	Pressing END sends the message, pressing c-END moves you to the message window.
Both Send	Pressing either END or c-END sends the message.
Add Text	(The default) Pressing END moves you to the message window. Pressing c-END sends the message.
Both Add Text	Pressing either END or c-END moves you to the message window.

Pressing END in the message window always sends the message.

zwei:*filter-alist-sort-predicate* *Variable*

Profile Option: Add Supersedes and Comments fields when retransmitting a message.

Controls the ordering of user-defined filters in the filter selection and creation menus. The choices are:

Alphabetically	The default. User-defined filters appear in the menus sorted by name.
Chronologically	User-defined filters appear in the menus sorted in the order in which they are defined. (That is, the first filter defined (loaded) into the world is at the top of the list.)

zwei:*indent-forwarded-msgs* *Variable*

Profile Option: Indent text of forwarded messages.

Controls whether a message that you forward is indented 4 spaces from the left margin, like messages in replies.

The choices are:

Yes	The forwarded text is indented 4 spaces from the left margin.
No	The default. The text is flush with the left margin, as in the original.

zwei:*reformat-forwarded-msgs* *Variable*

Profile Option: Reformat headers of forwarded messages.

Controls what happens to the headers of messages you forward. The choices are:

Yes	The default. The headers are pruned and reformatted.
No	The headers are left as in the original message. If you are forwarding a message to show someone something about its headers, you should set this option to No before forwarding the message.

zwei:*selected-dangerous-zmail-commands* *Variable*

Profile Option: Commands which will ask permission.

Controls the list of commands considered dangerous when typed from the keyboard. This option depends on the setting of "Ask permission before executing time consuming and/or destructive commands" (**zwei:*ask-before-executing-dangerous-zmail-commands***). If "Ask permission ..." is **selective**, you can select the commands that you want to have query you before execution from this list. If "Ask permission ..." is **all** or **none**, this option does nothing.

The commands in the list are:

zwei:com-zmail-save-mail-file

c-X c-S

zwei:com-zmail-kill-sequence
c-X K.

zwei:com-zmail-quit
Q from the keyboard.

zwei:com-zmail-save-all-mail-files
S from the keyboard.

zwei:com-zmail-expunge-sequence
Expunge from anywhere.

zwei:com-zmail-map-over
Map Over from anywhere.

zwei:*too-many-msgs-query-threshold* *Variable*
Profile Option: Maximum number of messages which may be mapped over without asking permission.

Controls how large a collection of messages you can operate on automatically without confirming the command. The default is 20.

9.2.4. New or Renamed Zmail Commands

- A new command, Change Mail File Options (m-X), has been added to Zmail. This command allows you to edit a mail file's options using the CVV window that, previously, was only available by clicking on [File options] in Zmail's profile editor.

- Enable Saves (m-X) has been renamed to Enable Saves for Buffer (m-X).

- Disable Saves (m-X) has been renamed to Disable Saves for Buffer (m-X).

- E (Expunge) is now implemented by **zwei:com-zmail-expunge-sequence** (it was **zwei:com-zmail-expunge**).

  S (Save) is now implemented by **zwei:com-zmail-save-all-mail-files** (it was **zwei:com-zmail-save**).

  The old function names will continue to work in case you have assigned them to key bindings in your `zmail-init` file.

9.2.5. Zwei Undo Facility Available From Zmail

The new Zwei Undo facility is available from Zmail.

You can undo (and redo) changes in any Zmail buffer, including the current message buffer, the body of each draft, and the header of each draft. Each buffer has separate undo and redo histories.

The simplest operation of the Undo facility is to undo the most recent change to the editor buffer. Go to a buffer, type something in, delete it, and then press

c-sh-U. The deletion is undone. Now press c-sh-R. You're back where you started. Keep pressing c-sh-U. Previous changes to the buffer are undone. You can keep doing this until the buffer is returned to its original state.

You can still use the Undo and Redo Zmail (m-X) commands as you always could, to undo and redo top-level Zmail commands, but now you can also undo and redo editing commands.See the section "Zwei Undo Facility", page 27.

9.3. Improvements to Zmail in Genera 7.2

9.3.1. Improvements to Zwei:preload-zmail

- A new preload request, **:edit-all-mail-files**, has been added that preloads all mail files listed in your profile. This request takes no arguments.

- The preload request **:find-file** has been renamed **:edit-mail-file** and the request **:examine-file** has been renamed **:examine-mail-file** for consistency with the recently added c-X commands. The old names for these preload requests will continue to work indefinitely.

- Previously, requests with no arguments had to be given as a list argument to **zwei:preload-zmail**. Now, they can also be given as symbols. In other words,

```
(zwei:preload-zmail '(:edit-all-mail-files))
```

  and

```
(zwei:preload-zmail :edit-all-mail-files)
```

  are now identical.

9.3.2. The End Key Can Now Be Customized in Zmail

When you are composing a message with Zmail, pressing END while you are positioned in the headers window moves your cursor to the message text window. If you wanted to send the message immediately, pressing END a second time was necessary. Now, pressing c-END from the header window sends the message immediately. From the message window, pressing END sends the message as before.

You can control the behavior of the END key with a Profile Option, "Effect of END key in *Headers* window." See the variable **zwei:*draft-editor-end-key-treatment***, page 139.

9.3.3. Destructive Commands Now Ask for Confirmation

Because the Save, Expunge, Quit, and Map Over commands can destroy information, and because they can be time consuming if your mail files are large, S, E, [Expunge], Q, and [Map Over] now ask for confirmation before proceeding. This prevents your accidently starting a Save, for example, by pressing S when you meant D from the keyboard. You can turn off this query in your Zmail profile by

setting the option "Ask permission before executing time consuming and/or destructive commands" to No. See the variable
zwei:*ask-before-executing-dangerous-zmail-commands*, page 138.

9.3.4. Converting Mail Files to Kbin Format Now Renames Them

When you convert a mail file to KBIN format, Zmail now renames it for you if appropriate: If the mail file's pathname extension reflects its format, Zmail automatically renames it to KBIN. For example, if your mail file is named `KJones.babyl`, Zmail renames it to `KJones.kbin`. No attempt is made to automatically rename files named `babyl.text`. If your mail file is named `babyl.text`, you should rename it yourself to `mail.kbin`. Do this by clicking on the pathname.

See the section "Converting Existing Mail Files to KBIN Format" in *Communicating With Other Users.*

9.3.5. Miscellaneous Improvements to Zmail

- Starting a new mail message, forwarding a message, and replying to a message and then aborting right away without making any additions or changes to the draft message no longer leave an empty draft cluttering up the Drafts list. This is particularly valuable for those times when you press the `M`, `R` or `F` key by mistake.

- You can now click `m`-Left on the name of a Lisp object in a Zmail message to edit the definition of the object.

- Expunging a mail collection now disables background message parsing requests so they no longer cause errors.

- `c-m-Y` and `c-m-sh-Y` now work in Zmail.

10. Changes to Files and Streams in Genera 7.2

10.1. New Features in Files and Streams in Genera 7.2

10.1.1. New Coroutine Stream Function for Genera 7.2

The new function **sys:open-coroutine-stream** combines and replaces the functionality of the following functions:

sys:make-coroutine-input-stream

sys:make-coroutine-output-stream

sys:make-coroutine-bidirectional-stream

sys:open-coroutine-stream *function* &key *(:direction* **:input***)* *Function*
(:buffer-size **1000***) (:element-type* **'character***)*

Creates either input streams, output streams, or bidirectional streams, each with a shared buffer, depending on the argument given to *:direction.*

- Creating input coroutine streams:

  Give *:direction* the argument **:input** to create two coroutine streams, an input stream and an output stream, with a shared buffer. **sys:open-coroutine-stream** returns the input stream. The output stream is associated with a new stack group and the input stream with the stack group that is current when **sys:open-coroutine-stream** is called. **:tyi** messages to the input stream cause the new stack group to be resumed and *function* to be called from that stack group.

  For an example of a coroutine input stream: See the section "Coroutine Streams " in *Reference Guide to Streams, Files, and I/O.* Creating output coroutine streams:

  Give *:direction* the argument **:output** to create two coroutine streams, an input stream and an output stream, with a shared buffer. **sys:open-coroutine-stream** returns the output stream. The input stream is associated with a new stack group and the output stream with the stack group that is current when **sys:open-coroutine-stream** is called. Using the **write-char** function on the output stream causes the new stack group to be resumed and *function* to be called from that stack group.

  For an example of a coroutine output stream: See the section "Coroutine Streams " in *Reference Guide to Streams, Files, and I/O.* Creating two bidirectional coroutine streams.

Give *:direction* the argument **:io** to create two bidirectional coroutine streams. The input buffer of each stream is the output buffer of the other. One stream is associated with a new stack group and the other with the stack group that is current when **sys:open-coroutine-stream** is called. **sys:open-coroutine-stream** returns the stream associated with the current stack group.

Using the functions **read-char** and **write-char** on the stream returned by **sys:open-coroutine-stream** cause the new stack group to be resumed and *function* to be called from that stack group. The argument to *function* is the second stream created by **sys:open-coroutine-stream**. The first stream is the one returned. *function* should use **read-char** or **write-char** on the stream that is its argument. These functions resume the stack group in which **sys:open-coroutine-stream** was called. In this way *function* and the caller of **sys:open-coroutine-stream** communicate through the shared buffers; output from one function becomes input to the other.

function takes a single argument, *stream*, which is the "other end" of the stream returned to the caller by this function.

:direction can be **:input**, **:output**, or **:io**. To create input coroutine streams use **:input**; to create output coroutine streams use **:output**; and to create bidirectional coroutine streams use **:io**. These are the values accepted by **open** as specified in *Common Lisp: the Language*.

:element-type can be any element type acceptable to **open**.

:buffer-size is the size of the buffer of the intermediate buffer. The value should usually be set to the default size.

Note: If more than one argument to *function* is needed, use lexical scoping.

10.2. Incompatible Changes to Files and Streams in Genera 7.2

10.2.1. Obsolete Parameters for si:make-serial-stream

Note: The following parameters for **si:make-serial-stream** are now obsolete but are still supported; use the generic function indicated instead of the parameter:

:clear-to-send
: Use the **clear-to-send** generic function instead. See the method **:clear-to-send**.

:request-to-send
: Defaults to **t**. If you want to change this to **nil**, use the **setf-request-to-send** generic function to do so. See the method **:setf-request-to-send**.

:data-terminal-ready
: Defaults to **t**. If you want to change this to **nil**, use the **setf-data-terminal-ready** generic function to do so. See the method **:setf-data-terminal-ready**.

10.2.2. Lozenge Character

In Genera 7.2, the lozenge character (◊) is a reserved character in pathnames. This is an incompatible change from previous releases.

10.3. Compatibility Note: Files and Streams

10.3.1. Compatibility Note: open

Although the **open** is a Common Lisp function, the Genera implementation is different from the specification in *Common Lisp: the Language* (*CLtL*) in a number of ways:

- *CLtL* defines a fixed set of keywords for **open**: **direction**, **element-type**, **if-exists**, and **if-does-not-exist**. The Genera implementation accepts additional keywords.

- *CLtL* says that the default **:element-type** for **open** is **zl-user:string-char**. In the Genera implementation, the default **:element-type** is **character**.

  CLtL says that **:element-type** acepts the following types: **zl-user:string-char**, (**zl-user:unsigned-byte** *n*), **zl-user:unsigned-byte**, (**zl-user:signed-byte** *n*) **zl-user:signed-byte**, **character**, **zl-user:bit**, (**mod** *n*), and **:default**. Specifically:

Element Type	*Status*
string-char	Suppored by Genera (but is not the default).
character	Supported by Genera (and is the default).
unsigned-byte	Not supported by Genera.
(**unsigned-byte** 8)	Supported by Genera.
(**unsigned-byte** 16)	Supported by Genera.
(**unsigned-byte** 32)	Supported by Genera for FEP files only.
(**unsigned-byte** *n*)	Not supported by Genera (except for indicated special cases)
signed-byte	Not supported by Genera.

(**signed-byte** *n*)	Not supported by Genera.
bit	Not supported by Genera.
(**mod** *n*)	Not supported by Genera (due to a bug).

- *CLtL* says that the only valid values of the keyword **:direction** are **:input**, **:output**, **:io**, **:probe** and **:direct**. The Genera implementation accepts a number of other values for this argument, such as **:in** and **:out**, and device-specific values such as **:block**.

11. Changes to Networks in Genera 7.2

11.1. Incompatible Changes to Networks in Genera 7.2

11.1.1. zl:site-name is Obsolete in Genera 7.2

The variable **zl:site-name** is obsolete as of Genera 7.2. The variable is still present but there is a compiler warning when you use it. The preferred source of the same information is in the variable **net:*local-site***. The site's name can be obtained as follows:

```
(send neti:*local-site* :name) => :SCRC
```

You can convert uses of **zl:site-name** as follows:

Former Programming Practice:

```
(when (eq zl:site-name :acme)
  (load "apricot:>smith>cerebrum-server"))
```

Preferred Programming Practice:

```
(when (eq (send neti:*local-site* :name) :acme)
  (load "apricot:>smith>cerebrum-server"))
```

11.2. New Features in Networks in Genera 7.2

Netbooting, the ability to boot a world from a remote machine and run it without having the booted world-load file on the local disk, has been added to Genera 7.2.

The Namespace Editor now has a Dynamic Window interface.

Sync-link gateways, which permit use of Symbolics machines as gateways between Symbolics sites, have also been added.

For more detailed information on these features: See the section "Genera 7.2: Operations and Site Management", page 176.

11.3. Improvements to Networks in Genera 7.2

11.3.1. Terminal Program Offers Some Dynamic Window Features in Genera 7.2

The Terminal program offers some dynamic window features. First, the window is scrollable, so you can scroll forward and backward over the history of input and output that has appeared on the screen. Second, you can mark a region of the screen and do something to it, such as: enter the marked region as input, save it on the kill ring, or hardcopy it.

The Terminal program does not use presentation types, nor does it support features of dynamic windows that use the SUPER or META keys. For example, m-W, m-V, and m-SCROLL do not work in the Terminal program.

Marking and Using Regions:

The marking features are available with the CONTROL key pressed down. If you press CONTROL the mouse documentation line shows which commands are available. Typically you first mark a region of the screen and then do something with that region. To mark a region, position the cursor at the beginning of the region of interest. Press c-LEFT and move the cursor to the end of the region of interest. The marked region is underlined. Press c-RIGHT for the *Marking and Yanking Menu*, which lists the things you can do with the region.

Entering a Region as Input:

You can position the cursor over a single word (not separated by spaces) and press c-MIDDLE to enter that word as input. If you want to enter a longer command that is separated by spaces and lines, mark the region, click c-RIGHT for the menu, and choose [Yank Marked Text].

Using the scroll bar:

The scroll bar on the left-hand side of the screen allows you to scroll backward and forward. Any cursor motion or graphics happens relative to the current viewport.

When you are using the TELNET or CTERM protocol, all input and output history is saved, and you can access it by scrolling. Most terminal connections to UNIX hosts use TELNET, and connections to VAX/VMS hosts use CTERM.

Note that when you are using the 3600-LOGIN or SUPDUP protocol, or emulating a VT100 terminal, only one screenful of input and output is saved, so you cannot scroll backward or forward. Most terminal connections between two Symbolics computers use 3600-LOGIN.

11.3.2. Terminal Program Offers Connection Keywords in Genera 7.2

In most cases you need only enter the name of the host to the "Connect to host" prompt in the Terminal program. However, there are optional keywords that let you further specify some aspect of the connection. These keywords include:

:Login protocol
: The name of the protocol to use to interpret output from the remote host. If this keyword is not supplied, a protocol is chosen automatically by the Generic Network System. You can enter any protocol for LOGIN service defined on your host. Some examples are: TELNET, SUPDUP, 3600-LOGIN, CTERM. The HELP key lists the LOGIN protocols defined on your host.

:Connection Protocol
: The name of the protocol to use to establish the connection. If this keyword is not supplied, a protocol is chosen automatically

by the Generic Network System, and it will be a protocol for LOGIN service. The HELP key lists the connection protocols defined on your host. You can spedify a protocol for a service other than LOGIN if you want to debug that server.

:Echo — If yes, echo all characters locally. If no, let the remote host echo the characters. The default is no.

:Overstrike — If yes, when the host outputs a backspace and you type another character in its place, the second character overstrikes the first. This behavior is similar to that of a printing terminal. If no, the backspace erases characters instead of overstriking them.

:Terminal Simulator — Specifies the name of a terminal you wish to emulate. The choices are: VT100, Ambassador, IMLAC, and Glass TTY.

:Wallpaper File — The pathname of a file to which output should be sent. This is sometimes called a journal file. By default there is no wallpaper file.

11.4. New Documentation for the Domain System

Here is new documentation for the Internet Domain System:

11.4.1. The Internet Domain System

The Domain System is a collection of specifications and procedures which implement the DOMAIN protocol, which is commonly used on the ARPA Internet. This particular protocol deals extensively with naming. It was created to address several problems.

One major problem that the Domain System addresses is the management of the ever growing number of hosts on the Internet. When there were only a few hundred hosts, it was reasonable to keep a master file of hosts in a central location to be copied across the network periodically. As more and more hosts were registered, the Internet administrators found that they wanted to separate the hosts into smaller administrative units. Information about these hosts would then be maintained locally. As a result, the domain system places these hosts in a tree-structured administrative system.

The second major problem that the Domain System attempts to address is the difficulty encountered when sending mail between different networks. Each network has a different naming scheme. These different naming schemes have hindered the interconnection of various networks. The Domain System attempts to allow connections between networks as diverse as ARPAnet, CSnet, BITnet, UUCP, Symbolics Dialnet, and others.

For instance, in the past, mail addresses looked like:

- *user%host*.CSnet@CSnet-Relay.ARPA
- *random-host!uninteresting-host!host!user*@UCBVAX.ARPA
- adi/*user%host*.BITnet@WISCVM.ARPA

When addresses are automatically generated by various mailers, the results can be combined to make truly horrendous addresses.

If all the hosts involved are using the domain system, all these mail addresses may be viewed as:

- *user@domain*

Symbolics follows the domain specification found in Request for Comments (RFCs) available from the Network Information Center, SRI International. We also use the same specification over Chaosnet.

The current (as of November 1987) relevant RFCs, including RFC number, author, title and, when relevant, document each RFC replaces are: RFC1035 (Mockapetris) *Domain Names - Implementations and Specifications* (Obsoletes RFCs 882, 883, 973), RFC1034 (Mockapetris) *Domain Names - Concepts and Facilities* (Obsoletes RFCs 882, 883, 973), RFC1033 (Lottor) *Domain Administrators Operations Guide*, RFC1032 (Stahl) *Domain Administrators Guide*, RFC1031 (Lazear) *Milnet Name Domain Transition*.

11.4.1.1. How the Domain System is Structured

Domains are administrative entities. There are no geographical, topological, or technological constraints on a domain. The hosts in a domain need not have common hardware or software, nor even common protocols. Most of the requirements and limitations on domains are designed to ensure responsible administration.

The domain system is a tree-structured global name space that has a few top-level domains. The top-level domains are themselves subdivided into domains. These domains can be further subdivided into yet more domains, and so on.

The administration of a domain requires controlling the assignment of names within that domain and providing access to the names, addresses, and list of valid services to users both inside and outside the domain.

The top-level domains are:

- **GOV** Government
- **EDU** Education
- **COM** Commercial
- **MIL** Military

- **ORG** Organization (an "other" category)

- **NET** Network administrative entities

Temporarily, the top-level domains also include:

- **ARPA** The current ARPA-Internet hosts

Additionally, the English two-letter codes identifying a country according to the International Standards Organization (ISO) Standard for *Codes for the Representation of Names of Countries* (ISO 3166, International Standards Organization, May 1981) can be used as top-level domains.

Sufficiently large companies can qualify for their own top-level domain. As of this writing, no company has attempted to qualify for a top-level domain.

11.4.1.2. How Domain Names Are Structured

The structure of a domain name is formally prescribed. Domains names are printed with each level of the domain name separated by a period. The Domain system knows nothing about hosts or sites, it deals only with names. The order of appearance in a domain name goes from the most specific to the most encompassing. For example, one of the Symbolics domain names is:

```
SCRC.Symbolics.COM
```

Based on the structure of the name, we can surmise that SCRC is the particular domain within Symbolics, and indeed it corresponds to a site in the Symbolics Namespace. Continuing, we deduce that Symbolics is the name of the company that has a domain name of `Symbolics.COM`, and that COM is the top-level domain for commercial organizations. It is equally possible that SCRC is the name of a host. There is no way to tell *from the name* what SCRC is.

A host named "Rocky" in the Aerospace department at Whatsamatta University might have the domain name of:

```
Rocky.Aero.Whatsamatta.EDU
```

Looking from the most general to the most specific, this host is a part of the EDU domain. More specifically, it is a part of the domain Whatsamatta.edu. Within the domain Whatsamatta.edu, there is another domain—Aero.Whatsamatta.EDU. At this point, there is nothing to tell us if Rocky is a domain or a host in a domain. We know that Rocky is a host, because it is stated above. But one must always be aware of this potential ambiguity when reading domain names.

A domain name is just a name. The naming convention requires that some authoritative entity agree that it will be responsible for providing information about some domain and will guarantee that the information provided will follow the domain conventions. There is nothing implicitly better, worse, different, or otherwise unusual about the number of segments in a domain name.

As a consequence of the above convention, periods are effectively reserved characters. The domain `Whatsamatta.edu` should not be referred to as `Whats.a.matta.EDU`. The latter is in an entirely different domain. A name must contain either a char-

acter, a numeral, a dash, an underscore, or some combination of these elements. Domain implementations are currently required to be case-insensitive.

11.4.1.3. Acquiring Domain Information

Domain information is acquired by Symbolics machines in one of two ways. The first way is to query other domains about information regarding domains for which they are authoritative. The second technique is to place information into the domain or namespace files of a site. In this section, we shall discuss the first method. The second method is discussed in the section "Additions for the Domain System", page 33.

Each domain on the Internet must have at least one *Name Server*, called the *Primary Name Server* and one *Secondary Name Server*. A Primary Name Server is a host that is guaranteed to give authoritative information about domains for which it is authoritative. A Name Server can be authoritative for a domain by virtue of being either Primary or Secondary for that domain. Note that the Server need not be under, over, in, or around the domain it's serving. Symbolics machines not on the Internet and running Dialnet need only have a Primary Name Server.

The job of a Name Server is to provide authoritative information about domains for which it is authoritative. The Name Server must provide the requested information to any host, whether the requesting host is local or foreign to the domain. If the Name Server does not know the requested information, it can request information in one of two ways: *recursively* or *non-recursively*. Non-recursive queries are also called iterative queries.

A user machine which does not know the address of a particular host will request information about this host from its local Name Server. If the local Name Server knows the information, it will provide this information to the user machine. This ends the query.

If the local Name Server does not know the information requested, it will then ask another Name Server. The second Name Server will inform the first Name Server of the answer to the query. The first Name Server will then provide the information to the user machine. This is known as a recursive query.

If a user machine requests a Name Server to give some information about a host and the Name Server determines that the information is not about a domain for which it is responsible, the Name Server will suggest another Name Server that is likely to have the answer. The user machine then queries the suggested Name Server. The user machine will receive an answer to its query or the user machine will be informed that no information is known. This is referred to as a non-recursive query.

Each host running Genera contains name resolver software. The Symbolics host requesting information does not necessarily query Name Servers on the network outside the site. This depends upon the configuration of the site and if the local Name Servers are responding.

If the domain server software is loaded on a Symbolics Lisp Machine, that machine will act as a *central name resolver* host for the site. Upon receiving a query, the central name resolver then requests the desired information from Name Servers on

the Internet. The returned information is shared with the requesting machine and is also stored in the local cache. If a second host requests the same information, it can quickly be returned by the central name resolver without making another network request.

The following information is added to the service attribute of a host object to make it a central name resolver:

```
DOMAIN CHAOS DOMAIN
```

If the resolver supports IP/TCP protocols, the following should also be added:

```
DOMAIN TCP DOMAIN
```

Symbolics machines can be used as Name Servers. This is true even if the machine is not connected to the Internet and running IP/TCP. All that is required is that the Name Server be capable of storing and providing information necessary to resolve Internet Domain Names. The Genera 7 software fulfills these requirements. Thus, the Symbolics machine acting as a Name Server need not have any special domain service attributes in its namespace.

More information about using Symbolics computers as Name Resolvers or as Name Servers can be found in "Internet Domain Names", volume 9, page 222 of the documentation set.

11.4.1.4. Registering Your Domain

Any organization desiring to register a domain name must contact the administrator of the containing domain. For companies under .COM, or institutions under .EDU, refer to the document *Domain Administrators Guide* (RFC 1032, Network Information Center, SRI International, November, 1987)

Those organizations that don't have a domain name, but are reachable through Symbolics Dialnet, are provided with an umbrella domain under which they can register their own domains. This domain is `Dialnet.Symbolics.COM`. For example, the host Twinkie at Yoyodyne Industries might have the domain name of:

```
Twinkie.Yoyodyne.Dialnet.Symbolics.COM
```

Registration as a Dialnet domain is accomplished by installing Dialnet, the store and forward mailer, and the domain system. The installation of these products is discussed in "Installing Symbolics Dialnet", volume 0, page 97 and in "Installing and Configuring the Mailer", volume 0, page 113 of the documentation set. After these are installed, send electronic mail to:

```
Customer-Reports@Riverside.SCRC.Symbolics.COM
```

To register, you should supply the following:

- the name of the company,
- the name of the domain under `Dialnet.Symbolics.COM`,
- the real name and the mail name of the individual responsible for the maintenance of the site,

- a voice telephone number to reach this individual, and
- the name and Dialnet address of the server running Dialnet.

You will receive return mail informing you that you have been registered.

Registration on both Internet and Dialnet is superfluous. If your machines run IP/TCP, that protocol will always be used in favor of Dialnet. IP/TCP has a higher priority than Dialnet because, among other things, it is a more efficient means of communication.

For more information, contact the administrator of the domain in which you wish to register.

11.4.1.5. The Domain System and the Namespace System

In many ways, the Domain System and the Namespace System attempt to solve the same problems. Both the Namespace System and Domain System attempt to deal with the issue of naming. Both systems deal with a collection of names that refer to a grouping of machines. In the case of the Namespace System, this collection is called the namespace. In the case of the Domain System, we shall refer to this as a domain or a subtree. However, it is important not to draw too close an analogy between the two.

It might appear that both systems map to "administrative entities". Actually, the Domain System returns attributes that are connected to names. The Namespace System goes beyond the Domain System in describing the hosts, users, printers, and networks within an entity known as a Site. There is nothing in the domain system that is equivalent to a Site. Humans make the connection between a name and an administrative entity like a site; the Domain System software deals *only* with names.

The major difference between the Symbolics Namespace System and the Domain System is that the space containing the set of all Namespace names is flat, whereas the Domain System is organized as a hierarchy. From one perspective, this hierarchy can be viewed as a tree-structured administrative hierarchy.

Any site which has Symbolics Lisp machines must use the Namespace System. Communication with other Symbolics machines within a site can occur without use of the domain system if the Symbolics machines are in the namespace system. Any site which wants to communicate with other sites must use the Domain System. If there are non-Symbolics machines at your site and you want to communicate with them via IP-TCP, you should run the Domain System. If you are running Dialnet and Genera, you must use the domain system.

The Domain System and the Namespace System appear to have information which overlaps. In point of fact, you cannot describe information via the Domain System that is also represented in the Namespace System. In other words, the Namespace System is *always* asked first, and it *always* wins any argument about the validity of any piece of data. If a query about a host that is mentioned in both the Namespace System and in the Domain System occurs, the information from the Namespace System will be used.

There is only one way of assuring that the information in the Namespace System and the Domain System don't conflict. This is accomplished by making a Symbolics Lisp machine the primary domain resolver for machines that are in the Symbolics namespace at your site. If this is done, the namespace information will be used to complete the domain information. If this is not done, data integrity will be compromised since you must manually update all host information in both the Namespace System and the Domain System at the same time.

If you have machines that are not part of the Symbolics namespace, then you can have a Symbolics machine serve as the piece of the domain tree that corresponds to the Symbolics namespace, and any other machine deal with other parts of the domain tree. In fact, this is the solution of choice. There is no useful way a machine can be a server for only part of a namespace/subtree. Note that nowhere in this discussion have we mentioned "site", only namespace.

You cannot have a partial representation of the hosts in the Namespace System and the remaining in a domain server elsewhere. Partial representation of information in one domain server and the rest in another domain server is also not allowed. The necessary and sufficient condition for confusion occurs when there is *not* a single authority for a block of names. Trouble arises when one server has one piece of the namespace/subtree and some other server has the rest of the namespace/subtree. *This restriction is not a characteristic of the namespace implementation nor of any domain implementation. Rather, it is a fact common to naming schemes.* If partial information were allowed, it is easy to see that problems would arise as soon as one server's information differed with another's. There *must* be an authoritative server in any naming scheme.

If your organization already has a domain resolver running on another system, you have two options:

- Move the domain resolver to a Symbolics machine.
- Create a new sub-domain containing the Symbolics machines with a Symbolics machine as a domain resolver.

11.4.1.6. Additions for the Domain System

Before the Symbolics Domain System process can start, it needs to know where it can find the root domain resolvers. Since the namespace system is already up and running, this information is put in the Namespace object that describes the site. The Namespace system of the local site needs to know its domain name when it interacts with the domain system. This is put in the Internet Domain Name attribute of the Namespace object in the namespace.

Other information for the Domain System is put in files which are of a standard format and can be read by any system which is running a Domain Server. One such file, the *launch file* is required by the Symbolics Domain System. It contains the minimum domain information and the location of other files containing extensive domain information. These other files are referred to as *domain files*, except for the dialnet registries which retain their existing names.

The actual format of the files is more flexible than what is described in the following sections, but for ease of explanation, it has been simplified for this discussion. For full documentation on the file format, see *Domain Names - Implementation and Specification* (RFC 1035, Network Information Center, SRI International, November 1987).

11.4.1.7. The Launch File

In the Genera Domain Resolver, we use a "launch" file which is formatted like the standard domain files. This file is `sys:site;Launch-Domain-Server.Text`.

A record takes one line and has three fields separated by one or more whitespace characters. Comments start with semicolons. The format of a record is:

<type> <domain> <meaning depends upon the type field>

- The first field in a launch file record is the *type* field.
- The second field is the *domain* field.
- There might be a third field. If so, its meaning depends upon the given *type* field.

There are four record types. The record type is determined by the *type field* of a record. The four record types are:

1. **Domain**: The *domain* field contains the name of the domain which this resolver serves.

2. **Primary**: The *domain* field is the name of a domain for which this resolver is a primary resolver. The third field contains the name of a domain file for this domain. The contents of this file is described below.

3. **Secondary**: The *domain* field is the name of a domain for which this resolver is a secondary resolver. The third field contains the Internet address of the primary resolver for that domain. Upon startup, the primary resolver is asked for all information in this domain.

4. **Dialnet**: The *domain* field contains `Dialnet.Symbolics.COM`. The third field contains the name of a dialnet registry. Typical names are `sys:site;private-dialnet-registry.lisp` and `sys:site;public-dialnet-registry.lisp`. This is a Symbolics extension to the Domain System that allows us to use a file format which is already in use at many Symbolics sites.

11.4.1.8. The Internet Domain File

The domain file is referenced by a Primary record in the launch file. This is a standard file which is read with its domain name taken from the domain field of the Primary record of the launch file. The Domain system specifies the format of this file. A typical name for a domain file is sys:site;Whatsamatta-Domain.text.

The first field in a domain file is the domain field. In many cases the domain name in this field is the name of a host. This field is optional. If it is omitted, it defaults to the value of the domain field in the previous record.

The second field is the class field. This is the class of protocol which is relevant for the given record. A class of IN means Internet, a class of CH means Chaosnet, and a class of DIAL means Dialnet.

The third field is the type field. The record types are described below.

Depending on the class and type of the record, there may be an RDATA field. The RDATA field is documented below.

Here are some notation conventions which can be used in this file:

.	A free standing dot refers to the current domain name.
..	Two free standing dots represent the null domain name of the root.
@	A free standing @ denotes the current origin. The origin is the name that is appended to names without trailing dots, and the value of "@".(See example below.)
\X	Where X is any character other than a digit (0-9), it is used to quote that character so that its special meaning does not apply. For example, you can use "\." to place a dot character in a label.
( )	Use parentheses to group data that crosses a line boundary. In effect, line terminations are not recognized within parentheses.
" "	A leading whitespace means the "last name read".

The domain system allows a label to contain any 8-bit character. Although the domain system has no restrictions, other protocols, such as SMTP, do have name restrictions. Because of other protocol restrictions, we recommend that you use only the following characters in a host name (besides the dot separator):

```
"A-Z", "a-z", "0-9", dash and underscore
```

All times are in units of seconds, and all integers are in decimal.

Domains referred to in this file are relative to the domain named in the main file. For an absolute domain name, append a dot to the end of the domain name. This means that if the launch file contains the following:

```
Primary     Symbolics.COM     sys:site;Symbolics-Domain.Text
```

Then the following translations occur in sys:site;Symbolics-Domain.Text:

File Entry	*Translation*
@	Symbolics.COM
Riverside.SCRC	Riverside.SCRC.Symbolics.COM
Vermithrax.SCH	Vermithrax.SCH.Symbolics.COM
Think.COM.	Think.COM

The current origin is Symbolics.COM. Each name following will have Symbolics.COM appended to it. Thus, Riverside.SCRC becomes Riverside.SCRC.Symbolics.COM. When Think.Com. is read, the trailing period states that this is the full domain name.

The record types which may appear in the domain file and are discussed here are:

SOA	Start Of Authority
NS	Name Server
MX	Mail Exchanger
A	Internet Address

SOA Start Of Authority. This record is traditionally the first record of the file. All other fields in this record are required. The *domain* field should contain the name of the domain.

Domain names in the zone files can be either of two types, absolute or relative. An absolute name is the fully qualified domain name and is terminated with a period. A relative name does not terminate with a period. The current default domain is appended to a relative name. The default domain is usually the name of the domain that was specified in the boot file that loads each zone.

The form of an SOA record is:

```
<name>  <class> SOA  <origin>  <person>  (
                <serial>
                <refresh>
                <retry>
                <expire>
                <minimum> )
```

The Start Of Authority record designates the start of a zone. The zone ends at the next SOA record.

<name> The name of the zone. A '@' is typically used here.

<class> The class field specifies the protocol group.

<origin> The name of the host on which the master zone file resides. Typically this is a relative name, as the zone will be owned by some host contained within it.

<person> The mailbox for the person responsible for the administration of this domain. This is formatted like a mailing address except the @-sign that normally separates the user from the host name is replaced by a dot. Typically this is a relative name.

RDATA field. The fourth field of all resource records. The RDATA field of an SOA record contains some of the information relevant to administration of the domain. In order of appearance here is the information contained in RDATA:

<serial> The serial number of the file is an unsigned 16 bit version number of the data in the file. This value should be manually incremented any time changes are made to the file.

<refresh> The refresh interval is an unsigned 32 bit number. It is the time interval, in seconds, before a secondary name server should check with the primary server to see if the data in the domain should be brought up to date.

<retry> The retry interval is an unsigned 32 bit number. It denotes the time interval, in seconds, after a failure that a secondary name server should wait before it attempts to refresh its information about the domain.

<expire> The expiration interval is a 32 bit number. It is the upper limit of time, in seconds, that a secondary name server should allow before the data in the domain can no longer be considered authoritative. At that point a new update should occur.

<minimum> The minimum interval is an unsigned 16 bit number. It is the minimum number of seconds that should be exported with any record from this zone (other than the SOA itself).

There should only be one SOA record per zone.

NS Name Server. This record form contains the name of a host which is acting as an authoritative name server for the domain. The form is:

<domain> <class> NS <server>

<name> The name of the zone. A '@' is typically used here.

<class> The class field specifies the protocol group.

<server> The name of a host which is acting as an authoritative name server for the domain.

MX Mail Exchanger. MX records specify where to deliver mail for a domain. The <preference> field in an **MX** record contains a 16-bit integer preference value. The

<host> field is the name of a host able to accept mail addressed to @i(domain) over @i(class) networks. The form of an MX record is:

<name> <class> MX <preference> <host>

<name> The name of the zone. A '@' is typically used here.

<class> The class field specifies the protocol group.

<preference> The preference field contains a 16-bit integer preference value.

<host> The host field host field is the name of a host that can accept mail addressed to domain over class networks.

There may be multiple MX records for a particular domain name. A preference value specifying the order a mailer should try multiple MX records when delivering mail may appear immediately before the host name. When mail is being sent to a domain, it may be sent to any host which has an MX record, but lower numbered MX records will be tried first. If a path already exist to that host the Symbolics mailer will use that path first.

An Address. An address record contains the Internet address of a host.

<host> <class> A <address>

<host> The name of a host.

<class> The class field specifies the protocol group.

<address> The Internet address of the host.

Domain System Examples

Some Examples

You should familiarize yourself with the following examples. Probably none of the scenerios are exact matches for your site's situation, but you will find that they cover most possibilities.

A Domain on the Internet

Example 1:

In this example, Whatsamatta University is a small school whose administration wants to create a network to connect its computer resources. Whatsamatta University heard of problems encountered by a neighboring school, Enormous State University, who did a wholesale implementation when it created its network. Whatsamatta University intends to grow its network in a step-by-step manner, to avoid having any of the same problems as Enormous State University.

Whatsamatta University requested and received a name of Whatsamatta.EDU from the NIC for its use on the Internet. The administration designated a Symbolics 3650, known as Bullwinkle, to act as the domain server for their site. Since most of the machines at Whatsamatta University will actually be part of one of the groups of computer used there, they decide to let the Computer Science department take care of administering Bullwinkle, rather than having a site containing only that one machine. The Computer Science (CS) department set up a domain called CS, under Whatsamatta.EDU. They set up their Site and Namespace information, called WhatsamattaU-CS and CS, respectively. (More about this below.) While in the process of getting ready to use Domain names, they give Namespace CS an Internet-Domain-Name property of CS.Whatsamatta.EDU. Since Host Bullwinkle is under control of the CS department, it lives in the CS namespace; thus it can be referred to as CS|Bullwinkle, or Bullwinkle.CS.Whatsamatta.EDU.

After making the appropriate hardware connections, the system maintainers load the domain system software on Bullwinkle. Then they created the following launch file:

```
; -*- Mode: Text -*-
;Boot file for name server (on Bullwinkle.Whatsamatta.EDU)
;
;This machine is the primary server for Whatsamatta.EDU.
;Please exercise caution when changing this file.
;

Domain  Whatsamatta.EDU
Primary Whatsamatta.EDU sys:site; Whatsamatta-Domain.Text
```

The next step is to create a domain file for Bullwinkle, named sys:site;\Whatsamatta-Domain.Text, containing the following:

```
; -*- Mode Text -*- ;

; Domain server info for Whatsamatta.EDU.  Please, exercise caution
; when changing this file.This file is referenced by domain server boot
; files (like  sys:site;launch-domain-server.text).
;
; Please increment the serial number field of the SOA
; record when you alter it.
    IN SOA Bullwinkle Postmaster.Bullwinkle (
                                1       ; Serial Number
                                86400   ; Refresh interval
                                1800    ; Retry interval
                                86400   ; Expiration period
                                1800    ; Minimum timeout
                                )
;
  IN NS Bullwinkle
  IN MX 10  Bullwinkle
```

The installation is a sucess.

Whatsamatta University's administration decides not to follow Enormous State University's lead and have all machines in the same domain. Instead they plan to follow the principles of the domain system. Each department will have it's own domain: the computer science department is CS.Whatsamatta.EDU, as mentioned above, and the film department's domain is JWard.Whatsamatta.EDU.

Whatsamatta University's Aerospace department has just set up a lab of Symbolics computers. The Aerospace department calls themselves Aero.Whatsamatta.EDU. The machine which will act as mail server, namespace server, file server, and domain server is named Rocky.Aero.Whatsamatta.EDU, with an internet address of [128.32.45.1]. The Aerospace department has convinced the history department to provide a secondary server for them. The history department is using a Symbolics computer named Peabody as their domain server.

The Aerospace department makes their sys:site;launch-domain-server.text file look like this:

```
; -*- Mode: Text -*-
;
;  Boot file for name server (on Rocky.Aero.Whatsamatta.EDU)
;
Domain   Aero.Whatsamatta.EDU
Primary  Aero.Whatsamatta.EDU sys:site;Aero-Domain.Text
```

They then create a file named sys:site;Aero-Domain.text which contains the following:

```
; -*- Mode Text -*-
;
; Domain server info for Aero.Whatsamatta.EDU.  This file is
; referenced by domain server boot files (like
; sys:site;launch-domain-server.text).   Please increment the serial
; number field of the SOA record when you alter it.
;
   IN SOA Rocky Postmaster.Rocky (
          1        ; Serial Number
          86400    ; Refresh interval
          1800     ; Retry interval
          86400    ; Expiration period
          1800     ; Minimum timeout
          )
;
IN NS Rocky
IN MX 10 Rocky
```

To signify that Peabody is a secondary server, they put the following line in Peabody's launch file:

```
Secondary Aero.Whatsamatta.EDU 128.32.45.1
```

They then make the following additions to the namespace:

- They add to the site object, WhatsamattaU-Aero:

```
ROOT-DOMAIN-SERVER-ADDRESS INTERNET 10.0.0.51
ROOT-DOMAIN-SERVER-ADDRESS INTERNET 10.0.0.52
```

They add to the namespace object, AERO:

```
INTERNET-DOMAIN-NAME AERO.Whatsamatta.EDU
```

Once the site maintainers have done this, they have to inform the parent domain (Whatsamatta.EDU) that they are up and running. The primary server for Whatsamatta.EDU is Bullwinkle.CS.Whatsamatta.EDU, run by the CS department, as described above. They therefore make the following modifications to the domain data file (Whatsamatta-domain.text) used by Bullwinkle:

```
Aero IN NS Rocky.Aero           ; Primary server
Rocky.Aero IN A 128.32.45.12    ; Rocky's address
Aero IN NS Peabody.History      ; Secondary server
```

This insures that any mail sent to Rocky.AERO.Whatsamatta.EDU from any site supporting domain addressing will arrive at the right place.

A New Domain on the Internet

Dr. Whoopie is a supplier who caters to many organizations on the Internet, and has his own link to it. His domain name is Whoopie.COM. Anticipating future expansion, Dr. Whoopie decides to use their domain name to specify a host's office location, even though now they have only one office on the network. This office is located in Cambridge, so they call it Cambridge.Whoopie.COM. There are two mail servers here: Duke, which is also the namespace server, and Honey. One of their better customers, the Aerospace department at Whatsamatta University, will act as their secondary server.

When Dr. Whoopie finishes making decisions about his site, he notifies the Internet administrators, giving them the appropriate information.

Dr. Whoopie's sys:site;launch-domain-server.text file looks like this:

```
; -*- Mode: Text -*-
;
; Boot file for name server (on Duke.Cambridge.Whoopie.COM)
;

Domain Whoopie.COM
Primary Whoopie.COM sys:site;Whoopie-Domain.Text
```

His sys:site;Whoopie-Domain.Text file looks like this:

```
; -*- Mode Text -*-
;
; Domain server info for Whoopie.COM.
; This file is referenced by domain server boot files (like
; sys:site;launch-domain-server.text). Please increment the
; serial number field of the SOA record when you alter it.
;
; If you don't know how, don't edit this file!
;
   IN SOA Duke.Cambridge Postmaster.Duke.Cambridge (
       1       ; Serial Number
       86400   ; Refresh interval
       1800    ; Retry interval
       86400   ; Expiration period
       1800    ; Minimum timeout
       )

;
IN NS Duke.Cambridge
IN MX 10 Duke.Cambridge
IN MX 10 Honey.Cambridge
Cambridge IN NS Duke.Cambridge
IN MX 10 Duke.Cambridge
IN MX 10 Honey.Cambridge
```

To provide a secondary server for Whoopie.COM, system maintainers add this line to the launch file for Rocky.Aero.Whatsamatta.EDU:

Secondary Whoopie.COM 192.35.28.4

They also make the following additions to the namespace:

- Add this to the site object:

```
ROOT-DOMAIN-SERVER-ADDRESS INTERNET 10.0.0.51
ROOT-DOMAIN-SERVER-ADDRESS INTERNET 10.0.0.52
```

  Add this to the namespace object:

```
INTERNET-DOMAIN-NAME Cambridge.Whoopie.COM
```

Now all mail sent to Duke.Cambridge.Whoopie.COM or Honey.Cambridge.Whoopie.COM from any site supporting domain addressing will arrive at the right place.

12. Changes to the FEP in Genera 7.2

12.1. Incompatible Changes to the FEP in Genera 7.2

12.1.1. FEP IDS Commands Renamed in Genera 7.2

The IDS-related commands have been renamed as World-related commands, but boot files with the old command names will continue to work, albeit with a warning that the command name is obsolete. Users should edit their boot files at their convenience to bring them up to date.

- Add IDS File has been changed to Add World File. See the section "Add World File FEP Command" in *Site Operations*.

- Clear IDS Files has been changed to Clear World Files. See the section "Clear World Files FEP Command" in *Site Operations*.

- Find IDS Files has been changed to Find World Files. See the section "Find World Files FEP Command" in *Site Operations*.

- Show IDS Files has been changed to Show World Files. See the section "Show World Files FEP Command" in *Site Operations*.

12.1.2. Load World Changed Incompatibly to Interact with Netbooting

The FEP Load World command has been changed incompatibly to interact with netbooting. See the section "Netbooting" in *Genera 7.2 Software Installation Guide*. In previous releases, it was an error if you attempted to boot an IDS world from your local disk and the parents of that world were not present. In Genera 7.2, if the parent worlds are all located in the FEP filesystem of an enabled netboot server, the parent worlds are netbooted. Symbolics is very interested in user feedback on this change, and on netbooting in general.

12.1.2.1. Netbooting IDS Worlds

There is no limit on using netbooting with IDS worlds. If the Netboot command in your boot file identifies an IDS world on a netboot server, all the parents of that world are also netbooted, provided they are on the same server.

You can also combine loading an IDS world from your local disk with netbooting. If you are accustomed to using a standard world for your site with your own custom IDS built on top of it, you can continue to do so. In this case, however, use the Load World command and specify the pathname of the IDS world on your local disk in your boot file. Do not use the Netboot command in this case.

Note: Not having all parents of an IDS world on the local disk was once an error, but is now interpreted as a request to netboot the missing parents.

Two conditions must be met to use this technique:

- You have an IDS world on your local disk, but not all its parents.
- All the remaining parent world loads are available on the same enabled netboot server.

In this case, the Load World command in the boot file loads the IDS world normally and searches the local disk for its parents. When some parent is not found, netbooting is initiated to boot that and all remaining parent worlds. When any parent world on the netboot server is deleted, you must, of course, create a new IDS on your local disk. You can netboot the new parent worlds and then create the new IDS.

Note: If there is no enabled netboot server on the local subnet, the Start command will start Lisp and Lisp will wait forever for a netboot server. No error is returned. If your site does not support netbooting, you will have to use `h-c-FUNCTION` to halt the netboot process and get back to the FEP and boot from the local disk.

12.2. New Features in the FEP in Genera 7.2

12.2.1. New FEP Commands in Genera 7.2

A number of new FEP commands have been added in Genera 7.2.

- See the section "Autoboot Delay FEP Command".
- See the section "Netboot FEP Command" in *Genera 7.2 Software Installation Guide.*
- See the section "Set Prompt FEP Command" in *Site Operations.*
- The FEP command Set Monitor Type has replaced the command Set Monitor-type for machines with the G208 FEP EPROM.
- See the section "Set World-to-netboot FEP Command" in *Genera 7.2 Software Installation Guide.*

The IDS-related commands have been renamed as World-related commands, but boot files with the old command names will continue to work, albeit with a warning that the command name is obsolete. Users should edit their boot files at their convenience to bring them up to date.

- Add IDS File has been changed to Add World File. See the section "Add World File FEP Command" in *Site Operations*.

- Clear IDS Files has been changed to Clear World Files. See the section "Clear World Files FEP Command" in *Site Operations*.

- Find IDS Files has been changed to Find World Files. See the section "Find World Files FEP Command" in *Site Operations*.

- Show IDS Files has been changed to Show World Files. See the section "Show World Files FEP Command" in *Site Operations*.

12.2.2. New FEP Command Documentation in Genera 7.2

New FEP command documentation has been added to Genera 7.2. All FEP commands have been documented. The documentation includes an indication of the source of each FEP command. Some FEP commands are available only on certain machine models, or on machines with certain FREP EPROMs, and other FEP commands are loaded by scanning a particular FEP overlay (flod) file. See the section "FEP Commands" in *Site Operations*.

12.2.3. Genera 7.2 Supports Color Consoles

Genera 7.2 supports color consoles, in addition to monochrome consoles. To use a color console, all you need is the correct hardware.

Booting a Machine With A Color Console

When you boot the Genera 7.2 on a color console, the initial Lisp Listener appears on top of the color screen. You can operate the Lisp Listener just as you would the Lisp Listener on a Symbolics monochrome monitor. If you need help using the Lisp Listener see: *Genera Users Guide* in the Genera 7.2 documentation set.

If you want to see the mouse documentation line in your color screen applications, use the Set Screen Options command. This command is under the section entitled Screen Options for Mapped Color Screen, and asks:

```
Status line present: Yes No
```

Click on Yes to include the mouse documentation line.

12.2.3.1. FEP Commands for the Color Console

You can issue the following commands to the FEP. These commands enable you to use, or change the use of, your cad buffer.

If you have v127 overlay files, or G206 FEP Eproms, you can use these FEP commands:

Color — If your system has both color and monochrome consoles, this command switches you to the color console.

This command is available only on color console systems with V127 or G206 FEP EPROMs.

You must either warm boot or cold boot after issuing this command.

Monochrome
If your system has both color and monochrome consoles, this command switches you to the monochrome console.

This command is available only on color console systems with V127 or G206 FEP EPROMs.

You must either warm boot or cold boot after issuing this command.

Set Color Monitor-type *monitor-type*
Identifies the color monitor to the FEP.

This command is available only on color console systems with FEP EPROMs V127 or G206.

If you have G208 FEP Eproms, you can use these FEP commands:

Set Console {color | monochrome}
Sets the console to either color or monochrome.

This command is available only on color console systems with FEP EPROM G208.

Set Monitor Type {color | monochrome}
{Philips | Moniterm | Amtron | Tektronix | Sony}

Identifies the console type.

This command is available only on systems with FEP EPROM G208.

This command replaces Set Monitor-type and Set Color Monitor-type for these systems.

12.2.3.2. Color Console: Troubleshooting

Some programs that work properly on a monochrome console will fail when run with a color console. The most common failure occurs in programs using the **:bitblt** or **:bitblt-from-sheet** messages, or programs that access the screen array directly with **bitblt**.

The element type of the color console's screen array is (**unsigned-byte** 2), while that of a monochrome console is **bit**. If you use **:bitblt** or **:bitblt-from-sheet**, the

source or target array must have the same element type as the screen array. The most common symptoms of this are "array access out of bounds" errors and compressed images, which appear to have every other bit on the X access missing.

The following changes will make failing programs work:

- Use the window messages, rather than using **bitblt** directly on the screen array.
- Use **tv:make-sheet-bit-array** or **tv:sheet-array-type** functions to make arrays to use with the **:bitblt** operation.
- Replace the **:bitblt** operation with a **:draw-1-bit-raster** operation, when this is appropriate.

Problems with Old Windows

The system will not attempt to move windows from a monochrome console screen to a color console screen or vice-versa. This means that if you save a world on a machine with a monochrome console and then boot it on a color console machine, the system puts aside all old windows, and it creates windows for system programs on the color screen. It is not possible to expose any of the old windows once you load the new world.

The preferred behavior for user applications is to create a new window on the new screen when needed. When you create windows with **dw:define-program-framework** or **tv:add-select-key**, this is done automatically. To fix your program:

- Use **dw:define-program-framework** or **tv:add-select-key** to select your window, if appropriate.
- Replace static allocation of windows, or use **defresource** with **tv:defwindow-resource**.

If you must do it yourself, before selecting the window, check to see if its screen is exposed. If the screen is not exposed, create a new window on some exposed screen.

If your application requires a unique window, you must move its window to an exposed screen before selecting it. To move a window, you can send the **:set-superior** message to the window. An example of a window treated in this way is the system ZMail window.

12.2.4. Correction of Disk ECC Errors

Several tools are available for correcting disk Error Correction Code (ECC) errors. These tools are **destructive in their operation** and should be used with care, particularly in correcting ECC errors in LMFS files.

ECC errors occur when a block or a series of blocks is written incorrectly to memory or to disk.

The Error Correction Code can be thought of as an extension to parity checking. Parity checking requires adding an additional bit and allows detection of single-bit errors. ECC requires appending more than one bit, but allows the detection of errors in more than one bit, and the correction of errors in at least one bit.

Memory ECC errors are generally an indication of hardware problems, but disk ECC errors may result from hardware problems, such as media defects, or from software problems.

This discussion covers software tools for disk ECC errors. In most cases, these tools can correct errors where there is no media defect. If these tools fail, there may be a media defect or other hardware problem. (Two known hardware causes of disk ECC errors are having an unterminated Ethernet cable, and halting the machine while it is writing to the disk.)

ECC errors can occur anywhere on the disk, that is, anywhere in the FEP File System. Wherever the error occurs, be sure to write down all disk information, such as unit, cylinder, surface, and sector. You may need this information to use the tools. If the error has occurred in a LMFS partition, there will be additional information, stating which partition and file the error occurred in.

Disk ECC errors to LMFS partitions are different from errors in other FEP files. For one thing, the destructive possibilities are greater, as you can **lose your entire LMFS**, not just a single file. The tool for correcting errors in LMFS partitions is called **lmfs:fix-file**.

Two tools, **si:fix-fep-file** and **si:fix-fep-block** are designed for use with other, non-LMFS FEP files.

If you have any doubts about using any of these tools, check with Symbolics first. The word **destructive** means just what it says. No one can bring back data lost through use of these tools.

In any case, you should be reluctant to delete and expunge a file with an ECC error. Deleting and expunging frees the blocks for use by other (new) files. The file receiving the faulty block may also report an ECC error. The integrity of the file system can be severely compromised, since it is uncertain when or where the error might reappear. The tools provided below treat the error and prevent the same block from damaging subsequent files.

12.2.4.1. Functions for Correcting Disk ECC Errors

lmfs:fix-file *pathname* *Function*

An ECC error in a LMFS partition is reported by the Debugger and looks like this:

```
>>Error: %DISK-ERROR-ECC during a %DCW-READ32 on unit 3.,
    Fatal ECC error,
    4. pending transfers associated with this disk event aborted.
    Word 0 of file header, record #o161470 of partition 3
         (File partition in FEP3:>lmfs3.file.1)
         For T:>sam>hacks>definitions.lisp.12
```

The recommended treatment for this error is to run **lmfs:fix-file** on the

machine that is getting the error. If the problem is in a directory, you should run the salvager, since some files may have been orphaned.

A directory is a file that contains an entry for each file residing in that directory. If the directory is damaged, entries may be lost. The corresponding files still exist, but do not appear in the directory listing. These files are considered orphans, and are often recoverable by the salvager.

lmfs:fix-file is a potentially dangerous tool and should be used with caution. It notifies the user of problems encountered and asks before taking corrective action, but it can destroy the file it is applied to. **Note:** If you do not understand the questions being asked or would like assistance with the use of this tool, please contact Symbolics Software Services.

Run **lmfs:fix-file** using the fully specified pathname mentioned in the error message:

```
(lmfs:fix-file "T:>sam>hacks>definitions.lisp.12")
```

lmfs:fix-file mentions problems it finds and queries the user before taking corrective action.

si:fix-fep-file *pathname* *Function*

si:fix-fep-file first reads all blocks in a file reported to have an ECC error and optionally performs a read/write test which will likely correct the error. The function can be used to remove these blocks from use and place them in the bad-blocks file.

Never use this function on a LMFS partition. For a similar function for use on LMFS partitions:See the function **lmfs:fix-file**, page 171. If there is a read error on a block, the user is asked whether to try a write-read test. If rewriting results in no read error, the function exits, because the ECC error has been eliminated.

If a read error persists, or if the user refuses the write-read test, there is a query as to what to do with the block. There are four choices:

1. Splice it out.
2. Replace it with a block of zeros.
3. Replace it with a block containing a copy of the bad block.
4. Delete the file.

The first three choices result in the bad block being put in the bad-blocks file.

Note that this means there is a way to put good (no-media-defect) blocks in the bad-blocks file. If there is an ECC error and the user refuses the rewrite test, thus not finding that the blocks have no defects, but chooses one of the first three options, the result is to write the blocks to the bad-block file.

A similar function, **si:fix-fep-block**, tests a single block.

si:fix-fep-block *unit cylinder head sector* &key *:force-write-test* *Function*

si:fix-fep-block first reads all blocks in a file reported to have an ECC error and optionally performs a read/write test which will likely correct the error. The function can be used to remove the blocks from use and place it in the bad-blocks file.

Never use this function on a LMFS partition. For a similar function for use on LMFS partitions:See the function **lmfs:fix-file**, page 171. If there is a read error, the user is asked whether to try a write-read test. If rewriting results in no read error, the function exits, because the ECC error has been eliminated.

If a read error persists, or if the user refuses the write-read test, there is a query as to what to do with the block. There are four choices:

1. Splice it out.
2. Replace it with a block of zeros.
3. Replace it with a block containing a copy of the bad block.
4. Delete the file.

The first three choices result in the bad block being put in the bad-blocks file.

Note that this means there is a way to put good (no-media-defect) blocks in the bad-blocks file. If there is an ECC error and the user refuses the rewrite test, thus not finding that the blocks have no defects, but chooses one of the first three options, the result is to write the block to the bad-block file.

A similar function **si:fix-fep-file** tests all the blocks in a file.

13. Changes to Tape and Disks in Genera 7.2

13.1. New Features in Tape and Disks in Genera 7.2

13.1.1. New Tape Distribution and Restoration Interfaces

Genera 7.2 features new interfaces for the Distribute Systems command and the Restore Distribution command, allowing you greater flexibility in distributing and restoring systems than did previous releases.

When you use the Distribute Systems command, and specify the keyword :Menu with the value Yes, you invoke the Distribute Systems frame. This frame enables you to specify distribution parameters for individual systems.

Likewise, when you use the Restore Distribution command with the keyword :menu with the value Yes, you invoke the Restore Distribution frame. This frame enables you to examine the list of systems and files on a distribution tape, select the ones you want, and restores them to your file system.

For detailed documentation on these new interfaces: See the section "Restore Distribution Frame" in *Genera Handbook*.See the section "Distribute Systems Frame" in *Genera Handbook*.

13.1.2. Genera 7.2 Supports New Disk

Genera 7.2 supports a new disk, the Maxtor XT4380.

13.1.3. Genera 7.2 Supports New Tape Drive

Genera 7.2 supports 6250 bpi tape drives.

13.2. Restore Distribution and Genera 7.0 Tapes

If you try to restore a Genera 7.0 distribution tape using the Restore Distribution command, the Systems to Restore pane displays the list of systems, plus one system named NIL. For example, you might see the following:

```
Systems to Restore

COMPILER-TOOLS-RUNTIME
COMPILER-TOOLS-DEVELOPMENT
FORTRAN-RUNTIME
FORTRAN
NIL
```

If you click on NIL, all the files in the Files to Restore pane disappear. If you

click on the name of a system, no change occurs. This is commensurate with the functionality of Restore Distribution in Genera 7.0, which did not allow you to select individual systems. (Restore Distribution in Genera 7.2 does allow you to select individual systems.)

This situation occurs because in Genera 7.0, there was no association between systems and individual files, hence the files all appear under the heading of "System NIL".

This does not affect the quality of the restoration; Genera 7.0 distribution tapes restore compatibly.

14. Genera 7.2: Operations and Site Management

Netbooting, the ability to boot a world from a remote machine and run it without having the booted world-load file on the local machine, is a new site-management tool in Genera 7.2.

There are also new CP commands for dealing with the namespace and a new Namespace Editor interface in Genera 7.2.

14.1. Netbooting New in Genera 7.2

A new site-management tool, netbooting, has been added to Genera 7.2. Netbooting is the ability to boot a world from a remote machine and run it without having the booted world-load file on the local machine.

The documentation includes some suggestions and rules for using netbooting, but Symbolics has a strong interest in hearing from users about how they have employed netbooting at their sites. See the section "Netbooting" in *Genera 7.2 Software Installation Guide.*

14.2. New Namespace Tools in Genera 7.2

Six new CP commands for dealing with the namespace and the Namespace Editor have been added in Genera 7.2:

- See the section "Add Service to Hosts Command" in *Genera Handbook.*See the section "Remove Service From Hosts Command" in *Genera Handbook.*
- See the section "Create Namespace Object Command" in *Genera Handbook.*
- See the section "Delete Namespace Object Command" in *Genera Handbook.*
- See the section "Edit Namespace Object Command" in *Genera Handbook.*
- See the section "Show Namespace Object Command" in *Genera Handbook.*

The commands to Add and Remove Service From Hosts allow you to add or remove a single services to a list of hosts or to all hosts, rather than adding the service to each host individually as in previous releases.

The other commands are part of a new interface to the Namespace Editor. The Namespace Editor now uses a dynamic window, and most commonly used namespace attributes have presentation types.

The Namespace Editor has

- Protection against entering incorrect data.
- Keyboard editing commands.
- A history of objects read in to the Namespace Editor and c-m-L and Next and Previous commands to allow you to move from one object to another. SCROLL and m-SCROLL also work.
- Prompting for attributes and portions of attributes.
- Completion.
- Clear indication that an object has been modified.
- An easy means of unmodifying an object. (m-~ unmodifies an object.)
- Object attributes always appear in the same order.
- Online Namespace Editor Help on what each object attribute does.
- Help for Services known by the local host.
- Error checking for unique Chaos, Internet, and DNA addresses within the local namespace and for the proper format for addresses.
- Checking for nicknames in use by other hosts in the local namespace.

See the section "The Namespace Editor" in *Site Operations*.

14.3. Linking Sites Through Sync-Link Gateways New in Genera 7.2

A light-duty synchronous-link gateway facility for using Symbolics machines to link two Symbolics sites is a new feature in Genera 7.2.

14.3.1. Sync-Link Gateways

This is a light-duty synchronous-link gateway facility for linking two Symbolics sites into a single network that can share files, send mail, copy worlds (albeit slowly), and exchange Converse messages using the Chaos and Internet (IP/TCP) protocols. (Sync-link gateways should be considered an additional Symbolics feature and not a full-service gateway.)

14.3.1.1. Hardware Requirements for Sync-Link Gateways

To use the sync-link gateways, the two sites must be connected by a dedicated synchronous link. In addition, the sites at each end of the link must have the following:

- Symbolics gate-array machine — Models 3610, 3620, 3630, 3650, or 3653 — to be used as a gateway. The processor load of being a gateway is light, approximately 5 percent, so this can be either a user or server machine.
- For gateway service at 9600 baud: a CSU/DSU modem connected to the **bulkhead** RS-232 port of the Symbolics machine by a male-to-female RS-232 extension cable.
- For gateway service at up to 56 kilobaud: a CSU/DSU modem plus an RS-232-to-V.35 interface converter such as the Black Box Model GA-IC221, or equivalent.

The alternative to the dedicated line — a switched dial line — is feasible, but not recommended.

The full hardware setup should be prepared before you enable the software.

14.3.1.2. Configuring Sync-Link Gateways

The sync-link gateway works by connecting two subnets through a third "subnet" that is the synchronous link. For example, assume that you have one site in Akron, one in Toledo and a trans-Ohio synchronous link. In fact, the two sites can be across the country or across the room. Only the two gateway systems are on the third subnet, but both must have addresses on that subnet. **Machines on all three of these subnets must have different Chaos and Internet addresses.**

Sync-link gateways can be configured such that there is one global network with one namespace or one network with two or more namespaces.

In general, if you have a site with several machines and are setting up a new satellite site with only a few machines, a single namespace is the better choice. On the other hand, if you have two long-standing sites with several machines, you will probably find it preferable to keep the namespaces separate. For more information: See the section "Bootstrapping Sync-link Gateways", page 180. Gateways can be configured for Chaos-only networks, or for networks with Chaos and Internet. Configuration of these two kinds of gateways is closely related, but discussed separately.

Configuring Chaos-only Sync-Link Gateways

If the gateway is to use the Chaos protocol only, use the Namespace Editor to edit the host objects of both gateway machines as follows.

First, add a Chaos address for the gateway machine on the link. **This is different from the host's Chaos address on the local subnet.** That is, the gateway machine has addresses both on the local subnet and the synchronous link, which is

also a subnet.

```
Address:  CHAOS 401
```

Now add a Peripheral: attribute for a Sync-Interface.

```
Peripheral: None Graphics-Tablet Kanji-Tablet Modem Pad
Sdlc-Interface Serial-Pseudonet Sync-Interface Other
  Unit: 1
  Baud: 300  600  1200  1800  2000  2400  3600  4800  7200  9600  19200  56000
  Chaos: 401
  Internet: an Internet address of the form A.B.C.D
  Clock-master: Yes  No
  Clock-constant: a decimal integer
  Rts-cts-protocol: Yes  No
```

The Chaos: attribute must have the same Chaos address number as the Chaos Address: attribute you just added.

The Internet: attribute is not used.

The Clock-master: and Clock-constant: attributes are not used. Timing is supplied by the dedicated synchronous link or the modem.

The Rts-cts-protocol: attribute should be **No** unless you are using a dial-up line, which is not recommended.

Configuring Sync-Link Gateways with Both Chaos and Internet

If the sync-link gateway is to use both the Chaos and Internet protocols, use the Namespace Editor to edit the host objects of both gateway machines as follows. You must have the Symbolics IP/TCP product to use the Internet protocol.

First, add a Chaos address and an Internet address for the gateway machine on the link. **These addresses are different from the host's Chaos and Internet addresses on the local subnet.** That is, the gateway machine has addresses both on the local subnet and the synchronous link, which is also a subnet.

```
Address:  CHAOS 401
Address:  INTERNET 128.81.1.1
```

Now add a Peripheral: attribute for a Sync-Interface.

```
Peripheral: None Graphics-Tablet Kanji-Tablet Modem Pad
Sdlc-Interface Serial-Pseudonet Sync-Interface Other
  Unit: 1
  Baud: 300  600  1200  1800  2000  2400  3600  4800  7200  9600  19200  56000
  Chaos: 401
  Internet: 128.81.1.1
  Clock-master: Yes  No
  Clock-constant: a decimal integer
  Rts-cts-protocol: Yes  No
```

The Chaos: attribute must have the same Chaos address number as the Chaos Address: attribute you just added.

The Internet: attribute must have the same Internet address number as the Internet Address: attribute you just added.

The Clock-master: and Clock-constant: attributes are not used. Timing is supplied by the dedicated synchronous link or the modem.

The Rts-cts-protocol: attribute should be **No** unless you are using a dial-up line, which is not recommended.

Now add a Service: attribute with the following service, medium, and protocol triple.

```
Service:  GATEWAY IP INTERNET-GATEWAY
```

Now add a User Property: attribute.

```
User Property: DEFAULT-INTERNET-GATEWAY hostname
```

In this case, *hostname* is the name of the host that serves as the Internet gateway to other networks. If you have no such system at your site, you do not have to supply this attribute.

14.3.1.3. Bootstrapping Sync-link Gateways

Before creating your sync-link gateways, you have to decide whether you want the two sites to share a single namespace or to have separate namespaces. In general, if you have a site with several machines and are setting up a new satellite site with only a few machines, a single namespace is the better choice. On the other hand, if you have two long-standing sites with several machines, you will probably find it preferable to keep the namespaces separate.

If you intend to have separate namespaces, all hosts you wish to have in communication must be running Genera 7.2 or a later release. If you intend to have a single namespace, only the gateway systems must be running Genera 7.2 or a later release.

Access is equal with either approach.

Bootstrapping Sync-Link Gateways with a Single Namespace

In general, if you have a site with several machines and are setting up a new satellite site with only a few machines, a single namespace is the better choice.

Here is the procedure to follow to set up gateways with a single namespace.

Both gateway machines must be running Genera 7.2 or a later release.

1. Create a host object for a gateway machine at the local site.See the section "Configuring Sync-Link Gateways", page 178.

2. Find a current world with namespace information in it. This is a world that has had Define Site or Set Site run on it, built after the gateway was configured in the namespace.

3. Load that world on the local gateway machine and boot it.

4. Take the same world to the new site and load it on the gateway machine there.

5. Boot the world at the new site.

6. Test the link using **zl:hostat**, specifying the octal Chaos address, not the host name. Use **tcp:send-icmp-echo**, specifying the Internet address in the format Internet|A.B.C.D, to test an Internet link.

7. Now you can make new worlds at the new site.

For an alternate technique: See the section "Bootstrapping Sync-Link Gateways with Separate Namespaces", page 181.

Bootstrapping Sync-Link Gateways with Separate Namespaces

If you have two long-standing sites with several machines, you will probably find it preferable to keep the namespaces separate.

Here are the constraints for this configuration:

- Both gateway systems must be running Genera 7.2 or a later release.
- All systems that you wish to be able to communicate across the gateways must also be running Genera 7.2 or a later release. Machines running earlier releases will be able to communicate within the local namespace, but when they are booted users will see notifications that the Global Network Name attribute is not recognized.
- Neither gateway server can be a primary namespace server in this configuration.

Here is the procedure to follow to set up sync-link gateways with separate namespaces.

1. Edit the host objects for the gateway systems at each site. See the section "Configuring Sync-Link Gateways", page 178. Reboot both systems or issue the Reset Network command on both systems.

2. Test the link using **zl:hostat**, specifying the octal Chaos address, not the host name. Use **tcp:send-icmp-echo**, specifying the Internet address in the format Internet|A.B.C.D, to test an Internet link.

3. Edit the Chaos and Internet network namespace objects at *both* sites and add a common Global Network Name to each object.See the section "The Global Network Name Network Attribute".

4. Reboot the namespace server for one site.

5. Use **neti:find-site** at that site, naming the second site. Now the first namespace knows about the other.

6. Save the world on the namespace server. Never go back to a world built before this time.

7. Reboot the namespace server at the other site.

8. Use **neti:find-site** at that site, naming the first site. Now both namespaces are aware of each other.

9. Save the world at the second site.

10. Optionally, add each site name to the other site's search list. Whether you do this depends on the amount of traffic you expect between the sites. If you do not do this, users will have to specify the other site's name when communicating with it.

11. Optionally, make each gateway machine a secondary namespace server for the other. Remember that the gateway machine cannot be a primary namespace server if there are separate namespaces. There are two ways of making the gateway machines secondary namespace servers:

    - Use the `Secondary Name Server:` namespace attribute. At the first site, name the second site; at the second site, name the first site. This will cause any namespace search to search of all namespaces known to each sync-link gateway namespace.See the section "The Secondary Name Server Namespace Attribute" in *Site Operations*.

    - Use the `Default Secondary Name Server:` host attribute for the two gateway machines. At the first site, name the second site; at the second site, name the first site. This will cause any namespace search to search only the two namespaces in the global network.See the section "The Default Secondary Name Server Host Attribute" in *Site Operations*.

For an alternate technique: See the section "Bootstrapping Sync-Link Gateways with a Single Namespace", page 180.

14.4. Console Serial I/O Port Caution

The console serial I/O port has architectural limitations that currently prevent the console processor from reporting errors during transmission. For this reason, Symbolics cannot recommend use of this port for applications that require a completely reliable data stream.

The console serial I/O port handles graphics and kanji tablets with no difficulty. (However, leaving the tablet puck or stylus on the pad makes the tablet send a

continuous stream of coordinates when connected to any serial port. Therefore, users should keep the puck or stylus off the tablet when not actually using it.)

Any user protocol that performs its own end-to-end reliability checking, such as Kermit or XModem, is also acceptable.

In its own operations, Symbolics uses LGP2 printers connected to the console serial I/O port with no difficulty, but some customers have had problems doing the same thing.

Improvements to the Print Spooler in Genera 7.2 detect LGP2 errors with much more reliability and reprint requests when errors are detected, but there is no guarantee that every request is correctly handled. The speed of screen dumps on printers connected to the console serial I/O port has improved, but screen dumps are still faster when the printer is connected to the bulkhead port.

Note: The cable for modems is **not symmetrical**. If it doesn't seem to work, swap ends and try again.

15. Changes to Documentation in Genera 7.2

Genera 7.2 changes are noted online in Document Examiner and in the printed documentation. Substantially changed books have been republished for 7.2:

Site Operations Book 0 includes information on the new Netbooting facility and a comprehensive, reorganized list of FEP commands.

Symbolics Common Lisp -- Language Concepts
Book 2A has been rewritten and reorganized, and now includes a large amount of new SCL/Common Lisp compatibility information.

Symbolics Common Lisp -- Language Dictionary
Dictionary entries in Book 2B now contain notes about every Symbolics Common Lisp extension and every incompatible function.

Programming the User Interface -- Concepts
Much new conceptual information and many new examples have been added to Book 7A.

Programming the User Interface -- Dictionary
Book 7B is now a comprehensive dictionary of objects. It includes documentation of new functions and previously undocumented old functions.

Genera 7.2 Reference Cards
The reference cards have been updated and expanded for this release.

In addition, a new subset of the documentation set is available in Genera 7.2. The *User's Guide to Symbolics Computers* has been expanded into four guides designed to help new users get started faster in Genera and experienced users find information more quickly:

Genera Concepts Book 1A provides a conceptual overview to orient new users and give them useful models to think about the environment.

Genera Workbook Book 1B is an introductory tutorial that leads a new user through the features of the environment. This is a revision of the previous Education Services tutorial and is intended for use with their courses as well.

Genera User's Guide
Book 1C is an introduction to the main subsystems and utilities of the Genera and instructions in their use.

Genera Handbook Book 1D is a volume of reference information in quick-reference format.

Although some books have not been republished for Genera 7.2, some information in them has been updated and added. When the online documentation for these books differs from the printed version, the online information is more up to date.

A new version of the *Software Installation Guide* is available both in print and, for the first time, online.

Note that volume 10 of the documentation set, the *System Index,* has not been republished for Genera 7.2. If you have a copy of this book from the Genera 7.0 or 7.1 documentation sets, page references to books republished in Genera 7.2 will no longer be accurate. You can get index information for newly published books from the following sources:

- The index in the back of each book.
- Document Examiner's Show Candidates and Show Overview commands.

16. Notes and Clarifications for Genera 7.2

16.1. Clarification on :printer-queue-control Service

The Genera 7.0 documentation had a typographical error in two sections: "Protocols Supported by all Symbolics Computers as Servers", and "Protocols Supported by all Symbolics Computers as Users". Those sections erroneously mentioned a service named **:printer-control-queue**. The service is actually named **:printer-queue-control**.

Thus, the correct information is:

Service	*Medium*	*Protocol*
PRINTER-QUEUE-CONTROL	CHAOS	PRINTER-QUEUE

16.2. New Documentation for Scheduler Function

Here is documentation for a previously undocumented function, **si:with-process-non-iteractive-priority**:

si:with-process-non-interactive-priority *(*&key *(:quantum-boost* **si:*process-command-initial-quantum***)) &body *body* *Function*

In order to allow you to type without interference while other processes are computing in the background, the Genera system raises the priority of your process while you type. This is called *interactive priority*. Often, you can see a process executing (or waiting) with an interactive priority by looking at the Processes display in Peek. A priority of 0+1 (base priority of 0, current priority of 1) almost always means that the process is currently interactive.

The process only executes with interactive priority while you edit, (for example, when you edit text, or when you use the input-editor). When you execute code, Genera lowers the priority to *non-interactive priority*, or the normal priority of the process.

The system uses interactive priority to allow your keystrokes to be processed in preference to computations - this makes typing seem smoother and more responsive. This also speeds up the response of mouse-sensitivity highlighting as you move the mouse. The system lowers your priority to

non-interactive priority when executing commands or forms so that the computation doesn't interfere with other computations (or with you trying to type to another window), so that, in principle, each activity gets a fair portion of the machine.

The system assumes that any input-editor command that has an input-editor accelerator will be used to edit text, and therefore, executes it at the interactive priority. The system makes the same assumption about presentation actions.

This can be a problem if you have a command that will execute for a long time, and will be executed at interactive-priority. It can use up a lot of the machine and interfere with other processes.

To solve this problem, you need to wrap a **(si:with-process-non-interactive-priority** () <body>) around the body of your form.

:quantum-boost Instead of just lowering the priority of this process, it also extends its quantum. Quantum is measured in units of 60ths of a second, and is a rough measure of how long the process holds the processor before yielding to another process of equal priority. The desired effect of *:quantum-boost* is to allow this process to continue to have high priority ("first among equals") for a short period of time after you using non-interactive priority. This allows short commands to execute quickly and makes the machine feel more responsive. The Default is **si:*process-command-initial-quantum***.

16.3. Note About the SELECT Key

We advise you not to use number keys for activity selection with the SELECT key. It is now reserved for possible future use as a numeric argument. See the function **tv:add-select-key** in *Programming the User Interface -- Dictionary.*

Index

=

=

`SELECT` = 111

=

A

A

Zmail Now Knows About `c-X` 136

A

Note About the SELECT Key 187
Pane State of **:accept-values** Pane 115
Simple Defstruct Accessor Functions Now All the Same 115
Acquiring Domain Information 153
Background Viewer Added to Document Examiner in Genera 7.2 93
New Commands Added to Mark Survey 136
Quick and Dirty Guide to Adding and Maintaining a Spell Dictionary 24
Additions for the Domain System 156
Add Output Field Command 52, 70
Add Output Subfield Command 52, 70
adjacency lookup 90
Compiled Advice 104
New Features for Advice 102
How Time Spent in Other Processes Affects Metering Results 64
Clearing Space After Installation 11
tv:key-test Function Again Available 111
Aligning Indentation 37
Hide All But Path to This Node Command 57, 73
Show All Node Descendants Command 57, 80
Dumping of **who-calls macros-expanded** Allowed 115
:allow-other-keys Is No Longer Valid 19
Genera 7.2 Allows Printing of TINY Character Styles 17
Terminal Program Allows You to Specify Protocol 149
Simple Defstruct Accessor Functions Now All the Same 115
Change in What Version of Component Systems Are Loaded When a System is Loaded 98
Metering Results Are Not Usually Repeatable 60
Free Standing Mail Buffers Are Now Retrievable 111
How Domain Names Are Structured 152
New Array Function in Genera 7.2 13
Destructive Commands Now Ask for Confirmation 142
You Can Now Customize Your `SELECT` Key Assignments 111

tv:key-test Function Again Available 111
Zwei Undo Facility Available From Zmail 141
More Sources Available in Genera 7.2 11

A

B

C

D

E

F F F

G

G

G

H

H

H

I

I

I

K

K

K

L

L

L

M

M

M

N

N N

O

O

P

P

P

Q Q Q

R R R

S S S

T T T

U

U

V

V

V

W W W

Y

Z