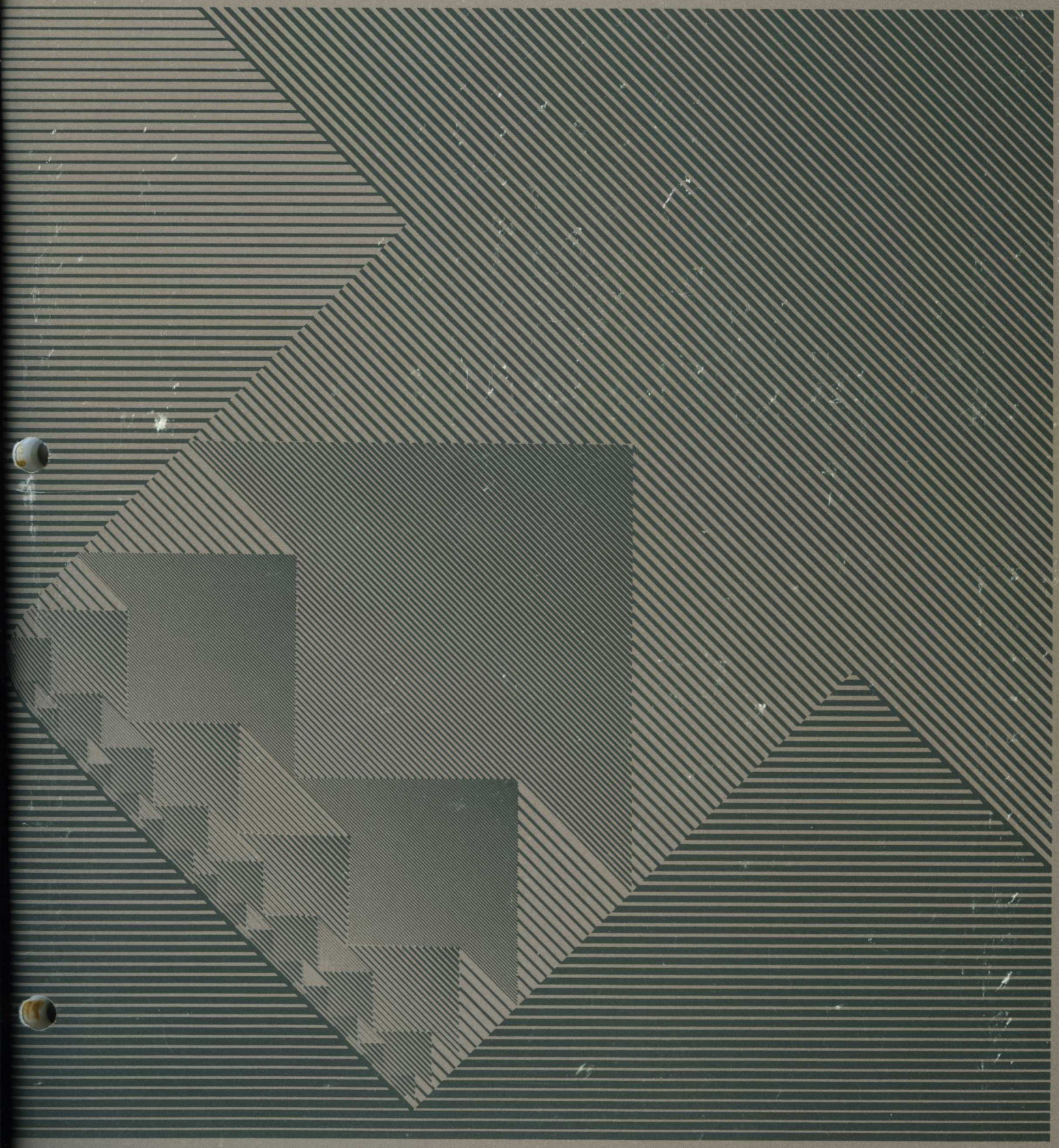
symbolics™

Release 4.0 Release Notes



symbolics ™

Release 4.0 Release Notes

January 1983

Cambridge, Massachusetts

Prepared by Symbolics, Inc.
Written by Jan Walker

This document corresponds to Release 4.0.

The information in this document is subject to change without notice and should not be construed as a commitment by Symbolics, Inc. Symbolics, Inc. assumes no responsibility for any errors that may appear in this document.

Symbolics, Inc. makes no representation that the interconnection of its products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting of a license to make, use, or sell equipment constructed in accordance with its description.

Symbolics' software described in this document is furnished only under license, and may be used only in accordance with the terms of such license. Title to, and ownership of, such software shall at all times remain in Symbolics, Inc.

Symbolics, Inc. assumes no responsibility for the use or reliability of its software on equipment that is not supplied or maintained by Symbolics, Inc.

Table of Contents

1. Introduction and Highlights

These release notes accompany Release 4.0. They describe changes made since System 210. The changes are organized in the following sections. Within each section, the material is organized in the following categories, in this order:

- incompatible changes
- new features
- improvements

Lisp Language and Compiler
: This section describes changes relevant to the Lisp language and compiler. The biggest changes are a new design for signalling and handling conditions and errors and a new design for pathnames.

Utilities
: This section describes changes in what any other computer would call the operating system and utilities. This includes the Debugger, the garbage collector, network support, and various system keyboard features. The most important changes are in the Debugger and garbage collector.

File System
: This section describes changes in the Lisp Machine file system. Most of the changes involved converting to use the condition signalling system.

Zmacs
: This section describes changes in the Zmacs editor. The biggest change for this release was reorganizing its internal data structures. Most of this change is visible only to those people writing editor extensions.

Zmail
: This section describes changes in the Zmail mail system. The most visible changes involve new terminology in menus and command names.

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

Operations and Site Management
: This section describes changes to the site configuration features of the system. Individuals who are responsible for the software at each site need to review these carefully.

You can find all the incompatible changes by reading the first part of each section. A complete list of changes appears in the Table of Contents.

The most important incompatible changes appear in the following sections:

Section	*Topic*
2.1.9, p. 17	Signalling and handling conditions
2.1.2, p. 7	Pathname changes
3.1.1, p. 53	Change in naming for compiled-code files

As in previous releases, many minor bugs have been fixed and performance in some areas has been improved. Only the more important or visible changes are mentioned here.

1.1 New microcode: 977

You must run Release 4.0 world loads with a microcode version of at least 977. Do not be alarmed if the microcode available is several versions newer than this; the microcode might have been updated after the notes went to press. Do not use a microcode version older than 977 however because the old world loads do not work with the new microcode and the new world loads do not work with the old microcode. **set-current-band** prompts you to set the current microcode as well as the current band, to help you keep consistent versions of the world load and the microcode.

1.2 Release numbering announcement

With this release, we are adopting a different procedure for numbering released systems. This is Release 4.0. Numbers from now on will be sequential, starting from this one.

Periodically we will issue an interim release of patches containing bug fixes and necessary improvements. Patch releases will also be numbered sequentially. For example, the first patch release for Release 4.0 would be Release 4.1, and so on.

The herald message now displays the release number; it does not display any system that is an unmodified part of the release. Thus you no longer see system numbers for System, Zmail, Microcode, and so on. Systems with unreleased patches and systems that were not part of the release (for example, Cube, Print, Macsyma, or user-defined systems) do appear in the herald.

A new function is available for determining the release version and status of any particular release. See section 3.2.2, p. 59.

1.3 Notation Conventions

The keys, like SHIFT or META, whose tops have black lettering are modifier keys, designed to be pressed in combination with other keys. They do not themselves transmit characters. Their combinations are shown hyphenated to remind you to press them at the same time as the associated key, not before.

The keys, like X or SYSTEM, whose tops have white lettering all transmit a character. Combinations of these keys are meant to be pressed in sequence, for example, SYSTEM L means to press the SYSTEM key, release it, and then press the L key.

The CTRL and META key combinations are abbreviated with c- and m-; the SUPER, HYPER, and SHIFT keys with s-, h-, and sh-, respectively. For example, the combined keypress META-X is pronounced "meta x" and written as "m-X".

This document uses the following notation conventions:

Appearance in document	*Representing*
send, chaos:host-up	Printed representation of Lisp objects in running text.
RETURN, ABORT, c-F	Keyboard keys.
SPACE	Space bar.
`login`	Literal type-in.
`(make-symbol "foo")`	Lisp code examples.
function name *arg1 arg2*	Syntax descriptions of definitions.
Undo, Tree Edit Any	Command names in Zmacs and Zmail appear with initial letter of each word capitalized.
Insert File (m-X)	Extended command names in Zmacs and Zmail. Use m-X to invoke one.
[Map Over]	Menu items.
(mouse-R)	Mouse clicks; L=left, M=middle, R=right.

Mouse commands use notations for menu items and mouse clicks in the following ways:

Square brackets delimit a mouse command; slashes (/) separate the members of a compound mouse command. The notation indicates which button to click only when that differs from the standard. For a single menu item, always click left. For example, the following two commands are exactly the same:

[Previous]
[(mouse-L) Previous]

For a compound command, always click right on each menu item except the last, where you click left. For example, the following two compound commands are exactly the same:

[Map Over / Move / Hardcopy]
[(mouse-R) Map Over / (mouse-R) Move / (mouse-L) Hardcopy]

For all other cases, the notation shows explicitly which button to click. For example:

[Map Over / (mouse-M) Move]

Some more examples:

- Suppose you are to click right on menu item [Map Over], then click right on menu item [Move], then click left on menu item [Hardcopy]. The notation is:
  [Map Over / Move / Hardcopy]
- Suppose you are to click left on menu item [Previous]. The notation is:
  [Previous]
- Suppose you are to click right on menu item [Map Over], then click middle on menu item [Move]. The notation is:
  [Map Over / (mouse-M) Move]
- Suppose you are to click right on menu item [Previous]. The notation is:
  [(mouse-R) Previous]

2. Lisp Language and Compiler

2.1 Incompatible changes

2.1.1 Reimplementation of hash tables

All hash tables are now implemented as instances of flavors. As a result, many of the functions related to hashing can now be replaced with messages. This was done to make it easier to write generic functions that work with any kind of hash table. The old functions will continue to work indefinitely.

The new hash table flavors are **si:eq-hash-table** and **si:equal-hash-table**. The hash table flavors accept a set of messages with names that are very similar to the old functions. The new messages take the same arguments as the functions that they replace (with the exception of the hash table argument because that is now implicit).

Functions	*Corresponding Messages*
gethash, gethash-equal	**:get-hash**
puthash, puthash-equal	**:put-hash**
remhash, remhash-equal	**:rem-hash**
swaphash, swaphash-equal	**:swap-hash**
maphash, maphash-equal	**:map-hash**
clrhash, clrhash-equal	**:clear-hash**

For example:

Old: `(gethash key foo-table)`
New: `(send foo-table ':get-hash key)`

You can now use **make-instance** to create an instance of a hash table. In addition, the functions **make-hash-table** and **make-equal-hash-table** are still available. **make-instance** and these functions now both accept the same set of init options:

:area
:growth-factor
:rehash-before-cold (see section 2.1.1.5, p. 6)
:size

For example:

```
(setq newcoms (make-hash-table ':size 500.))
(setq newcoms (make-instance 'si:equal-hash-table ':size 500.))
```

In addition to the messages that correspond to the previous functions, hash table instances accept the following messages:

:modify-hash *key function* &rest *other-args* to **basic-hash-table** *Method*

This method combines the actions of **:get-hash** and **:put-hash**. It lets you both examine the value for a particular key and change it. It is more efficient because it does the hash lookup once instead of twice.

It finds *value*, the value associated with *key*, and *key-exists-p*, which indicates whether the key was in the table. It then calls *function* with *key*, *value*, *key-exists-p*, and *other-args*. If no value was associated with the key, then *value* is **nil** and *key-exists-p* is **nil**. It puts whatever value *function* returns into the hash table, associating it with *key*.

```
(send new-coms ':modify-hash k foo a b c) =>
(funcall foo k val key-exists-p a b c)
```

:size to **basic-hash-table** *Method*

Returns the number of entries in the hash table, whether empty or filled. This means the amount of storage allocated, not the number of hash associations currently stored.

:filled-entries to **basic-hash-table** *Method*

Returns the number of entries in the hash table that have an associated value. This is the number of hash associations currently stored.

2.1.1.1 Loop iteration over hash tables

A new iteration path was added to **loop** to support iterating over every entry in a hash table.

```
(loop for x being the hash-elements of new-coms ...)
(loop for x being the hash-elements of new-coms with-key k ...)
```

This provides for *x* to take on the values of successive values of hash table entries. The loop runs once for every entry of the hash table. *x* could have the same value more than once, since it is the key that is unique, not the value.

The **with-key** phrase is optional. It provides for the variable *k* to have the hash key for the particular hash entry value *x* that you are examining.

2.1.1.2 Arguments to :map-hash

Like the **maphash** function, **:map-hash** takes one argument, which is a mapping function. In addition, **:map-hash** accepts extra arguments, which are passed on as arguments to the mapping function.

2.1.1.3 Dumping hash tables to files

You can dump hash tables that are flavor instances to files. Use **sys:dump-forms-to-file** (see section 2.2.29, p. 43) or whatever other dump function you need. The hash table flavors have **:fasd-form** methods.

2.1.1.4 Low-level internal issues

In previous releases, **equal** hash tables called **sxhash,** which returns 0 as the hash value for objects with data types like array, stack group, or pathname. As a result, hashing such structures could degenerate to the case of linear search. The new implementation solves this problem by using a different function (**si:equal-hash** instead of **sxhash**) that uses **%pointer** to define the hash key for such data types. This means that some of the hash keys in **equal** hash tables are based on a virtual memory address. Hash tables that are at all dependent on memory addresses are rehashed when the garbage collector flips.

Hash tables can now be accessed correctly from multiple processes; the hash table methods take care of proper locking.

2.1.1.5 Init option: :rehash-before-cold

:rehash-before-cold is an option that causes **disk-save** to rehash this hash table if its hashing has been invalidated. (This is part of the before-cold initializations.) Thus every user of the saved band does not have to waste the overhead of rehashing the first time they use the hash table after cold-booting.

For **eq** hash tables, the hashing is invalidated whenever garbage collection or band compression occurs because the hash function is sensitive to addresses of objects and those operations move objects to different addresses. For **equal** hash tables, the hash function is not sensitive to addresses of objects that **sxhash** knows how to hash but it is sensitive to addresses of other objects. The hash table remembers whether it contains any such objects.

Normally a hash table is automatically rehashed "on demand" the first time it is used after the hashing has become invalidated. This first **:get-hash** operation is therefore much slower than normal.

The **:rehash-before-cold** option should be used on hash tables that are a permanent part of the system, likely to be saved in a band saved by **disk-save**, and to be touched by users of that band. This applies both to hash tables in the Lisp system itself and to hash tables in user-written subsystems that are saved on disk bands.

2.1.2 Pathname changes

Many changes have been made to the specification, definition, operation, and performance of pathnames:

- A new mechanism for rationalizing case in pathnames (section 2.1.4, p. 9) to support writing system-independent code.
- A new canonical type mechanism for pathnames (section 2.1.5, p. 10).
- A new caller interface to pathnames (section 2.1.6, p. 14).
- A new mechanism for mapping wildcard pathnames (section 2.2.4, p. 27).

These sections supersede much of the material on pathnames in *Lisp Machine Manual*, section 22, p. 376*ff.*

2.1.3 Validity checking

Pathname code now does more checking of the validity of pathname components. Some previous code that contained hidden bugs will not work anymore. For creating intermediate pathnames, use **:new-default-pathname** instead of **:new-pathname** (section 2.1.6, p. 14).

Logical pathnames now check to be sure that their directory components have valid translations defined for them. This check is performed when a **:translated-pathname** message is sent. (It provides a proceed-type that allows you to specify a translation for a logical directory that does not have one.) It no longer just uses the logical directory name (*Lisp Machine Manual*, p. 388).

2.1.3.1 Merging pathnames from different host types

Most problems with merging Lisp Machine file system pathnames with those of other systems have been fixed. Some problems remain:

- Length restrictions in pathname components. The canonical type system alleviates this problem for type components.
- Character set differences in general. For example, you still cannot copy to VMS a file any of whose components contains a hyphen.

- Case information disappears between UNIX and the Lisp Machine file system (or between any system that guarantees to maintain case information and one that does not).

2.1.3.2 Change to representation for nil pathname components

The goal of defaulting pathnames is to produce a pathname with no components that are **nil**. Sometimes **nil** components do occur. In the rare occasions when you do see displays of pathname strings that contain the character "ξ", it represents a **nil** component.

You can also type the character ξ (TOP L) to designate a null component in a file spec when that is necessary in order to use a default component. For example, suppose you want to supply a new type to be merged with the default offered by Find File in the editor. The following example would find the file f:>rloon>tank.for:

```
Find File: (Default is f:>rloon>tank.lisp)
*.for
```

2.1.3.3 Meaning of :unspecific clarified

In a pathname, a value of **:unspecific** for a pathname component now means that the component is meaningless in the pathname syntax of the host. The following table shows the conditions under which a pathname component can be **:unspecific**.

Component	*Conditions for* **:unspecific**
Host	Never
Device	Always **:unspecific** for logical pathnames and pathnames for Unix, Multics and the Lisp Machine file system.
Directory	Never (for currently supported hosts)
Name	Never
Type	Sometimes **:unspecific** on ITS (see section 2.1.3.5). Always **:unspecific** for UNIX and Multics file specs in which a name component is specified with no period.
Version	Always **:unspecific** for UNIX and Multics pathnames. Sometimes **:unspecific** for ITS (see section 2.1.3.5).

Its use is now limited to these cases.

2.1.3.4 Generic pathname changes

Generic pathnames (as returned by the **:generic-pathname** message to **fs:pathname**) now have **nil** as the type and version instead of **:unspecific**. This was done as part of clarifying the meaning of **:unspecific**.

fs:*known-types* now contains canonical type symbols rather than type strings. (These are the types that generic pathnames strip off.) (See *Lisp Machine Manual*, p. 382).

:bin
:lisp
nil
:qbin
:unspecific

2.1.3.5 ITS pathname merging

Pathnames for ITS have been changed so that it is no longer possible to have two ITS pathnames with the same meaning that differ in an ignored component. **fs:*its-uninteresting-types*** still

exists; it controls which types are ignored in favor of retaining version numbers. The following table summarizes the interaction of type and version components for ITS pathnames.

Type	*Version*	*Result*
supplied	omitted	type is retained, version is **:unspecific**
omitted	supplied	type is **:unspecific**, version is retained
"interesting"	supplied	type is retained, version is **:unspecific**
"uninteresting"	supplied	type is **:unspecific**, version is retained

2.1.4 Case in pathnames

The material in this section supersedes the *Lisp Machine Manual* section on pathname messages, p. 384.

The standard messages to pathnames for accessing components have all been changed so that components are interchangeable between hosts. The new forms and the old forms of the messages look the same but are compatible only in code that does not depend on the flavor of pathnames. The purpose of this change is to enable and encourage the writing of host-independent code.

The components of a pathname (directory, name, type, and so on) have two possible representations for case, *raw* (also called *native*) and *interchange*. The raw case representation keeps the case in whatever form is normal for that system (for example, lower case for UNIX, upper case for TOPS-20). Interchange representation is a format for manipulating pathname components in a host-independent manner. All pathname defaulting and cross-host translation functions use the interchange form of pathname messages.

All of the standard messages to pathnames (for example, **:directory**, **:name**) now return pathname components in interchange case rather than raw case. This was done to minimize disruption to existing code. The pathname flavors accept a new set of messages that return components in raw case.

Interchange case form	*Raw case form*
:device	**:raw-device**
:directory	**:raw-directory**
:name	**:raw-name**
:type	**:raw-type**

The interchange form of the message specifies the following effect:

Case of component	*Translated case returned*
System default	Upper case
Mixed case	Mixed case
Opposite to default	Lower case

Upper case was chosen as the interchange case because strings like "LISP", representing pathname components, appear in many programs. Either choice (upper or lower) would have been natural for some hosts and not for others.

This facility provides more features for dealing with pathname components independent of the case-sensitivity of file names of different hosts. The following table shows some examples for different host types.

Host	*Message*	*Applied to raw form*	*Returns interchange form*
UNIX	**:name**	"foo.bar"	"FOO"
	:name	"FOO.BAR"	"foo"
	:name	"Foo.Bar"	"Foo"
Lisp Machine file system	**:name**	"foo.bar"	"FOO"
	:name	"FOO.BAR"	"FOO"
	:name	"Foo.Bar"	"FOO"
TOPS-20	**:name**	"FOO.BAR"	"FOO"
	:name	"foo.bar"	"foo"
	:name	"Foo.Bar"	"Foo"

Note that the Lisp Machine file system appears not to follow the interchange case rules. This is because, for the Lisp Machine file system, case is usually maintained but is not significant ("foo", "Foo", and "FOO" are all the same). Thus any mixture of cases in a file name satisfies the "system default" condition and hence returns all upper case for the interchange form.

The following functions and messages now use interchange case.

fs:make-pathname
: Four new keyword options, named **:raw-directory** and so on, specify the components for the new pathname in raw form.

:new-pathname Four new keyword options, named **:raw-directory** and so on.

:new-default-pathname
: Four keyword options, named **:raw-directory** and so on.

In addition, new messages are available for manipulating the raw case form of pathname objects.

:new-raw-*xxx* Four new messages to pathname objects, one for each pathname component that can be a string:

:new-raw-device
:new-raw-directory
:new-raw-name
:new-raw-type

These have the same kind of function as the **:new-*xxx*** messages. See *Lisp Machine Manual*, p. 384.

The raw forms of the messages are provided for writing host-specific code or for manipulating several pathname objects known to be on the same host.

2.1.5 Canonical types in pathnames

A *canonical type* for a pathname is a symbol that indicates the nature of a file's contents. When you need to compare the types of two files, particularly when they could be on different kinds of hosts, you would compare their canonical types. (**fs:*default-canonical-types*** and fs:*canonical-types-alist* show the canonical types and the default surface types for various hosts.)

Some terminology:

canonical type A host-independent name for a certain type of file, for example, Lisp compiled code files or LGP font files. A canonical type is a keyword symbol.

file spec What you type when you are prompted to supply a string for the system to build a pathname object.

surface type The appearance of the type component in a file spec. This is a string in native case.

default surface type
Each canonical type has as part of its definition a representation for the type when it has to be used in a string. Default surface type is the string (in interchange case) that would be used in a string being generated by the system and shown to the user. (See **fs:define-canonical-type,** p. 13).

preferred surface type
Some canonical types have several different possible surface representations. The definition for the type designates one of these as the preferred surface type. It is a string in interchange case. ("Default surface type" implies "preferred surface type" when one has been defined.)

Each canonical type has a default surface representation, which can be different from the surface file type actually appearing in a file spec. **:lisp** is a canonical type for files containing lisp source code. For example, on UNIX, the default surface representation of the type for **:lisp** files is "L". (Remember, the default surface representation is kept in interchange case.) The surface type in a file spec containing lisp code is different on different systems, "LISP" for Lisp Machine file system, "l" for UNIX. You can find out from a pathname object both the canonical type for the pathname and the surface form of the type for the pathname by using the **:canonical-type** message (p. 12).

The following tables illustrate the terminology.

UNIX

Surface type	"l"	"lisp"	"foo"
Raw type	"l"	"lisp"	"foo"
Type	"L"	"LISP"	"FOO"
Canonical type	**:lisp**	**:lisp**	"FOO"
Original type	**nil**	"LISP"	"FOO"

Lisp machine

Surface type	"l"	"lisp"	"foo"
Raw type	"l"	"lisp"	"foo"
Type	"L"	"LISP"	"FOO"
Canonical type	"L"	**:lisp**	"FOO"
Original type	"L"	**nil**	"FOO"

To translate the type field of a pathname from one host to another, determine the canonical type, using the surface type on the original host. Then find a surface type on the new host for that canonical type.

Copying operations can preserve the surface type of the file through translations and defaulting rather than converting it to the surface form for the canonical type. For example:

```
(multiple-value-bind (ctype otype)
    (send p ':canonical-type)
  (send p ':new-pathname
        ':canonical-type ctype
        ':original-type otype
        ':name "temp-p"))
```

2.1.5.1 UNIX and VMS change

This change is mostly compatible with previous releases. The only change is for certain cases on VMS and UNIX. Previously, VMS and UNIX pathnames used to explicitly transform type strings so as to fake the behavior of canonical types. This had strange side effects. For example, it used to be impossible to refer to a UNIX file called foo.lisp because the pathname code would transform that into foo.l. These problems no longer occur.

2.1.5.2 New messages

A number of new messages and methods have been supplied for dealing with canonical file types.

:canonical-type to **fs:pathname** *Method*

The **:canonical-type** message determines the canonical type of a pathname and a surface representation for the type. It returns two values:

Value	*Meaning*
canonical type	This is either a keyword symbol from the set of known canonical types or a string (when the type component of the pathname is not a known canonical type). The string contains the type component from the pathname, in interchange case.
original type	This is **nil** when the type of the pathname is the same as the preferred surface type for the canonical type (see **fs:define-canonical-type,** p. 13). Otherwise, when the type differs from the preferred or default surface type, it is the original type in interchange case.

For example, for a UNIX pathname, sending the message **:canonical-type** to the following pathnames has these results:

Pathname	*Results from* **:canonical-type** *message*		
foo.l	**:lisp**	**nil**	Preferred surface type
foo.lisp	**:lisp**	"LISP"	Alternate surface type
foo.L	"l"	"l"	Not recognized
foo.LISP	"lisp"	"lisp"	Not recognized

Keep in mind that the **:canonical-type** message returns the type string in the interchange case rather than in the raw case.

:new-canonical-type *canonical-type* &optional *original-type* to **fs:pathname** *Method*

The **:new-canonical-type** message to a pathname returns a new pathname based on the old one but with a new canonical type. *canonical-type* specifies the canonical type for the new pathname. The surface type of the new pathname is based on the default surface type of the canonical type, unless the pathname already had the correct type.

When the pathname object receiving the message already has the correct canonical type, the surface type in the new pathname depends on the presence of *original-type.* When *original-type* is omitted, the new pathname type has the same surface type as the old pathname. When *original-type* is supplied, the surface type for the new pathname is

original-type. This assumes that *original-type* is a valid representation for *canonical-type*; if that assumption is not met, the *canonical-type* prevails and its default surface type is used.

canonical-type is a symbol for a known type, **:unspecific**, **nil**, or a string. Use a string for *canonical-type* to make pathnames with types that are not known canonical types.

The following examples assume that a pathname object for the file spec "vixen:/usr/jwalker/mild.new" is stored in **m**.

```
(send m ':new-canonical-type ':lisp) =>
#<UNIX-PATHNAME "VIXEN: //usr2//jwalker//mild.1">
(send m ':new-canonical-type ':lisp "LISP") =>
#<UNIX-PATHNAME "VIXEN: //usr2//jwalker//mild.lisp">
(send m ':new-canonical-type ':lisp "MSS") =>
#<UNIX-PATHNAME "VIXEN: //usr2//jwalker//mild.1">
(send m ':new-canonical-type "BAR" "BAR") =>
#<UNIX-PATHNAME "VIXEN: //usr2//jwalker//mild.bar">
(send m ':new-canonical-type ':lisp "lisp") =>
#<UNIX-PATHNAME "VIXEN: //usr2//jwalker//mild.1">
(send m ':new-canonical-type ':lisp nil) =>
#<UNIX-PATHNAME "VIXEN: //usr2//jwalker//mild.1">
```

fs:define-canonical-type *canonical-type default* &body *specs* *Special Form*

fs:define-canonical-type defines a new canonical type. *canonical-type* is the symbol for the new type; *default* is a string containing the default surface type for any kind of host not mentioned explicitly. The body contains a list of specs that define the surface types that indicate the new canonical type for each host. The following example would define the canonical type **:lisp**.

```
(fs:define-canonical-type :lisp "LISP"
  ((:tops-20 :tenex) "LISP" "LSP")
   (:unix "L" "LISP")
   (:vms "LSP"))
```

For systems with more than one possible default surface form, the form that appears first becomes the preferred form for the type. Always use the interchange case.

Define new canonical types carefully so that they are valid for all host types. For example "com-map" would not be valid on VMS because it is both too long and contains an invalid character. You must define them so that the surface types are unique. That is, the same surface type cannot be defined to mean two different canonical types.

Canonical types that specify binary files must specify the byte size for files of the type. This helps **copyf** and other system tools determine the correct byte size and character mode for files. You specify the byte size by attaching a **:binary-file-byte-size** property to the canonical type symbol. For example, the system defines the byte size of press files as follows.

```
(defprop :press 8. :binary-file-byte-size)
```

fs:find-file-with-type *pathname canonical-type*

fs:find-file-with-type searches the file system to determine the actual surface form for a pathname object. Like **probef**, it returns the truename for *pathname*. When no file can be found to correspond to a pathname, it returns **nil**.

canonical-type applies only when *pathname* has **nil** as its type component.

fs:find-file-with-type searches the file system for any matching file with *canonical-type*. For example, on a TOPS-20 host, this would look first for ps:<gcw>toolkit.lisp and then for ps:<gcw>toolkit.lsp:

```
(fs:find-file-with-type (fs:parse-pathname "sc:<gcw>toolkit") ':lisp)
```

If it finds more than one file, it returns the one with the preferred surface type for *canonical-type* (or chooses arbitrarily if none of the files has the preferred surface type).

If *pathname* already had a type supplied explicitly, that overrides *canonical-type*. You can ensure that *canonical-type* applies by first setting the type explicitly:

```
(fs:find-file-with-type (send p ':new-type nil) ':lisp)
```

System programs that supply a default type for input files (for example, **load**, **make-system**, **qc-file**) could use this mechanism for finding their input files.

:types-for-canonical-type *canonical-type* to **fs:pathname** *Method*

:types-for-canonical-type is the internal primitive for finding which surface types correspond to *type*. Normally you would not use this directly. See **fs:find-file-with-type** to determine what form of a pathname exists in a file system.

2.1.5.3 New options for :new-pathname

:canonical-type and **:original-type** are new keyword options to the **:new-pathname** method to **fs:pathname**. (See *Lisp Machine Manual*, p. 385.) These are equivalent to sending the **:new-canonical-type** message to the result of the **:new-pathname** message but they are faster and more convenient for specifying several components.

In addition, the **:type** option has been changed to take a symbol argument that is a canonical type as well as a string in interchange case.

:canonical-type is for host-independent applications; **:type** is available for when you are dealing with strings.

2.1.5.4 Correspondence of canonical types and editor modes

fs:*file-type-mode-alist* is an alist that associates canonical types (in the car) with editor major modes (in the cdr).

```
((:LISP . :LISP) (:SYSTEM . :LISP) (:TEXT . :TEXT) ...)
```

See also section 5.3.11, p. 93 for discussion of editor major modes.

2.1.6 Changes in the caller interface to pathnames

This section supersedes most of *Lisp Machine Manual*, section 22.4, p. 382. It describes changes to the caller interface for handling file specs that have been supplied by a user.

2.1.6.1 fs:*defaults-are-per-host* replaced

The new cross-host defaulting mechanism replaces the variable **fs:*defaults-are-per-host*** (*Lisp Machine Manual*, p. 381). This variable still exists but no longer affects pathname defaulting or any other aspect of system operation. The variable will be removed entirely in a future release.

2.1.6.2 fs:merge-pathnames supersedes fs:merge-pathname-defaults

fs:merge-pathnames *pathname* &optional *(defaults* ***default-pathname-defaults****) (default-version* **:newest***)*

fs:merge-pathnames supersedes **fs:merge-pathname-defaults** (*Lisp Machine Manual*, p. 382).

New programs should always use **fs:merge-pathnames**. **fs:merge-pathname-defaults** still exists but it tends to violate the rules for the meaning of **:unspecific** (section 2.1.3.3, p. 8). It did not use the type from the displayed default, which people found confusing.

More specifically, **fs:merge-pathnames** does not default types as explained in *Lisp Machine Manual*, p. 380. It defaults types the same way that it defaults names.

The following example shows the style now recommended for showing defaults to users.

```
(setq default (send x ':new-canonical-type ':lisp))
(format t "Default is ~A~%" default)
(fs:merge-pathnames (readline-trim) default)
```

2.1.6.3 Changes to values returned by pathname component messages

The messages that return components of a pathname now return somewhat different sets of values. The differences involve mostly which special symbols can be returned. Any messages that return strings now return them in interchange case (section 2.1.4, p. 9).

- The **:host** message returns a host object.
- The **:device** message returns one of the following:
  nil
  a string
  :unspecific
- The **:directory** message to a pathname now returns one of the following values:
  nil
  :root
  :wild
  a list of strings
  a list of strings and the following symbols
    :wild
    :wild-inferiors
    :relative
    :up

  It never returns a single string (as it used to in previous releases).
- The **:name** message returns one of the following values:
  nil
  a string
  :wild
- The **:type** message returns one of the following values:
  nil
  a string
  :unspecific (for ITS, see p. 8, UNIX, and Multics)
  :wild
- The **:version** message returns one of the following values:

:newest
number
:oldest
:unspecific (for UNIX, Multics, ITS)
:wild

2.1.6.4 New message for building pathnames: :new-default-pathname

:new-default-pathname &rest *options* to **fs:pathname** *Method*

:new-default-pathname returns a new valid pathname based on the one receiving the message, using the pathname components supplied by *options*. The components do not need to be known to be valid on a particular host. The method uses the components "as suggestions" for building the new pathname; it is free to make substitutions as necessary to create a valid pathname. It is heuristic, not algorithmic so it does not necessarily yield valid semantics. The heuristics used, however, seem to produce pathnames that match what many people expect from cross-host defaulting.

It always produces a pathname with valid syntax but not necessarily valid semantics. For example when it tries to map between a hierarchical file system and a nonhierarchical file system, it uses the least significant of the hierarchical components as the directory component. Sometimes this is not correct but in all cases it is syntactically valid. The main application for **:new-default-pathname** is in producing defaults to offer to the user.

Application notes: **:new-pathname** always does what its arguments specify; it never uses heuristics. Thus **:new-pathname** could signal an error in certain cross-host defaulting situations where **:new-default-pathname** would not have any problems. In general, user programs should be using **fs:default-pathname**, which sends **:new-default-pathname** as part of its operation.

2.1.6.5 Change to argument for fs:merge-pathname-defaults

The third argument to **fs:merge-pathname-defaults** used to be the type component of a pathname (*Lisp Machine Manual*, p. 382). It now accepts a canonical type keyword as well as a string (see section 2.1.5, p. 10.)

By the way, **fs:merge-pathname-defaults** has been superseded anyhow by **fs:merge-pathnames**. See section 2.1.6.2, p. 14. Programs that use **fs:merge-pathname-defaults** should really be converted to use **fs:merge-pathnames**.

2.1.6.6 make-pathname-internal replaced

Previous releases contained an internal function called **make-pathname-internal**. It has been removed and replaced with a message.

Old:
```
(make-pathname-internal host dev dir nam typ vrs)
```
New:
```
(send host ':get-pathname dev dir nam typ vrs)
```

2.1.6.7 New method: :system-type

:system-type to **fs:pathname** *Method*

The **:system-type** message returns the type of host that the pathname is intended for. This value is a keyword from the following set:

:its, :lispm, :multics, :tenex, :tops-20, :unix, :vms, :logical

This is the same set as returned by the **:system-type** message to a host object.

See also **fs:default-pathname**; it is not likely that you need to use this message directly.

2.1.6.8 New method: :sample-pathname

:sample-pathname to **si:pathname-host** *Method*

The **:sample-pathname** message sent to a host object requests a pathname for receiving messages. It returns a syntactically valid pathname with this host as the host component. This replaces the former practice of calling **fs:default-pathname** with only a host argument.

2.1.7 Relative pathname support

Relative pathnames are now supported for all host types that allow relative pathnames in their native syntax. In particular, the Lisp Machine file system now supports relative pathnames (section 4.2.1, p. 74.)

2.1.8 Character set name changes

The Control-Shift- characters are now encoded differently. c-sh-A is no longer a synonym for h-c-A; they are now distinct compound keystrokes.

This change is part of a clarification of policy on key naming and key meaning. In addition to the four modifier keys HYPER, SUPER, CTRL, and META, the SHIFT key is now a modifier key for letters when used in combination with one of the other modifiers. As before, the CAPS LOCK key is not a modifier key and is always ignored in compound keystrokes. Thus typing CTRL and A at the same time gives c-A; typing CTRL and SHIFT and A at the same time gives c-sh-A. Typing CTRL and SHIFT and / at the same time gives c-? (not c-sh-/).

The names for compound key strokes always show a letter as capitalized. This does not mean that you have to use the SHIFT key; use the SHIFT key as a modifier only when "sh-" appears in the key name.

In addition, printing names of characters now have case in them, for example, Roman-III. (Case continues to be ignored on input. Some new synonyms for existing characters are now accepted. In particular, names of the following form have new synonyms.

New	*Equivalent to*
#\c-sh-B	#\c-shift-B
#\mouse-L	#\mouse-L-1

The symbol **si:xr-character-names** has been removed. The names of the characters are in the table in SYS: IO; RDDEFS LISP.

2.1.9 Signalling and handling conditions

The mechanisms in the Lisp Machine for signalling and handling conditions have been redesigned. (The term *conditions* includes errors.) A new document, *Signalling and Handling Conditions*, explains the new procedure in detail. This new document supersedes most of Chapter 26 in the *Lisp Machine Manual*. The Debugger has been revised to use the new condition design.

2.1.9.1 Program compatibility and conversion issues

You have two choices in revising your program to use the new condition signalling. You can use the style of signalling that is mostly compatible with the old design and make minimum program changes or you can revise your program to take advantage of the power available from the new facility. (See the summary in section 2.1.9.2 and read *Signalling and Handling Conditions.*)

2.1.9.2 Strategies for converting a program to use signalling

In previous systems, **error-restart** and **condition-bind** were used for sophisticated error handling. These functions were the precursors to the new condition system. If your program used these functions, you need to first understand the new condition handling design and then rethink your implementation of error handling; no simple conversion strategy is possible.

System functions that used to return a string to indicate an error now signal an error. Some functions used to either return a string or trap to the Debugger, depending on what arguments you used. In the cases where they used to return strings, they now return error objects for compatibility. In the cases where they used to trap, they now signal an error condition.

Any program that is still checking for returned strings as an error flag needs to be revised. See the information about **errorp**, p. 19. In general, functions should not be "returning" errors but rather signalling them.

2.1.9.3 Incompatibility with Debugger

When you enter the Debugger, the variable **self** is bound to the condition object (because the error-signalling functions now use flavors). To access the value of **self** in some frame, select that frame and use the Debugger command `m-S`. At this time, Debugger evaluation occurs in the environment of the highest frame. In a future release, this will be changed to evaluate in the variable environment corresponding to the current frame.

2.1.9.4 Incompatibility with the previous design

This section discusses how to use the new design without doing a major overhaul of your program's error handling. Some functions work unchanged; some have changed the meaning of a few arguments. Several functions have the same name but different meaning.

Obsolete or incompatible features that must be changed

This section pertains to the following items:

:change-properties (to **fs:pathname**)
:delete (to **fs:pathname**)
:rename (to **fs:pathname**)
cerror
condition-bind
deletef
error-restart
errorp
ferror
fs:all-directories
fs:change-file-properties
fs:directory-list
fs:expunge-directory
fs:file-properties
open, the **:error** option
renamef
signal
sys:command-level
undeletef

:change-properties, :delete, and :rename

The **:change-properties**, **:delete**, and **:rename** messages to pathnames and streams have been incompatibly changed. They no longer accept an *error-p* argument. They always signal an error rather than returning an error. Define a handler to handle the error.

```
(condition-case ()
    (send fname ':delete)
  (fs:file-not-found nil))
```

cerror

The function **cerror** is obsolete. A version of **cerror** still exists and works compatibly only in simple cases without conditions or restart features. The following calls are compatible:

```
(cerror t nil nil ...)
(cerror nil nil nil ...)
```

All uses of **cerror** should be replaced by calls to the appropriate error signalling function. (**fsignal** might be appropriate in some cases.)

condition-bind

The syntax of the new **condition-bind** is the same as that of the old **condition-bind**. The names of all the error conditions have changed, however, so all calls to **condition-bind** must be updated. All handler functions must be revised to take arguments and return values in the new way.

error-restart

The name **error-restart** has been recycled. The new syntax is not compatible with the old one. Any program that uses the old **error-restart** must be modified to use the new syntax.

errorp

Several functions, including the following, used to take **:noerror** keywords or *error-p* arguments.

deletef
fs:all-directories
fs:change-file-properties
fs:directory-list
fs:expunge-directory
fs:file-properties
renamef
undeletef

These allowed the caller to request that a string be returned if an error occurred. These options still exist but now return an error object instead of a string. The function **errorp**, which was introduced in System 210 to help make this conversion easier, now checks for error objects rather than strings. These options are all obsolete; the modern way to deal with errors is by setting up a handler.

ferror

The arguments to **ferror** have changed; the former first argument no longer exists. Calls to **ferror** in which the first argument is **nil** are treated as if they had been converted to the new syntax. Calls to **ferror** in which the first argument is a symbol are obsolete and must be changed.

open, the :error option

The **open** function now interprets its **:error** option differently. This option controls what happens when any **fs:file-operation-failure** condition is signalled (see *Signalling and Handling Conditions*). The option has three possible values:

Value	*Meaning*
t	Signals the error normally. **t** is both the default and the recommended value.
nil	Returns the condition object.
:reprompt	Reprompts the user for another file name and tries **open** again. This behavior used to be the default in System 210. There it caused bugs because programs could not know that the file opened under these conditions was not necessarily the one that they requested. When you use this option, remember that the **:pathname** message sent to the stream finds out what file name was really opened.

t is the recommended value for this option. The others have been provided for compatibility with previous systems to aid in converting programs.

The alternative to **:reprompt** is to use **:error t** and set up a condition handler for **fs:file-operation-failure** that explains the condition and prompts the user.

signal

The syntax of the new **signal** is similar to that of the old **signal** but the meaning is different. All calls must be changed.

sys:command-level

The **sys:command-level** catch tag has been removed; use **catch-error-restart** instead.

Old version:

```
(*catch 'sys:command-level
   ...)
```

New version:

```
(catch-error-restart ((error sys:abort) ...)
  ...)
```

The package system now signals an error if it sees the symbol **sys:command-level** because the **sys** package is locked. (See section 2.1.12 on p. 23.) This should make old uses of **sys:command-level** very easy to find.

Functions that might require change

Programs containing the following might require some change, depending on the arguments involved.

time:parse
time:parse-universal-time
chaos:finger
chaos:whois

Some functions that used to return strings to indicate errors now signal errors normally. **time:parse, time:parse-universal-time, chaos:finger,** and **chaos:whois** are among the affected functions.

The **probef** function still returns **nil** if the file is not found (signals **fs:file-not-found**). It signals an error if anything else goes wrong (such as **sys:host-not-responding**). You can treat various other errors as if the file were not found by making the handler return **nil**. For example:

```
(condition-case ()
    (probef path)
  (sys:connection-error nil))
```

By writing this **condition-case** yourself, you can control precisely which conditions are reflected as a returned **nil** value and which should really be treated as an error.

Functions that work the same without change

Programs containing the following should work without change after being recompiled.

check-arg
check-arg-type
catch-error
err
errset
error

The old Maclisp **err** and **errset** functions still exist (see the *The Lisp Machine Manual,* p. 449). **err** is officially an obsolete function; it signals its error using **ferror**. **errset** just uses **catch-error**. We want to discourage you from using **errset** and **catch-error** because they can hide bugs as well as catch expected errors. See also section 3.1.5, p. 54.

The **error** function continues to try to be compatible with the old Maclisp **error** function wherever possible. It is usually possible when the first argument is not a condition name or a condition object. When it can, the compiler warns about incompatible uses of the **error** function.

2.1.10 Changes to fquery options

The default for the **:list-choices** option to **fquery** has been changed from **nil** to **t**. This affected the functions **y-or-n-p** and **yes-or-no-p** in particular (see p. 48).

The **:condition** option to **fquery** is now obsolete; in its place is the new option **:signal-condition**. Its application is the same as that for **:condition**. Basically it is a way to intervene and provide an answer to a query without asking the user.

The default for **:signal-condition** is **nil**. When its value is **t**, the **fquery** function signals an **fquery** condition with proceed type of **:choice** before prompting the user. Any handler can invoke the **:choice** proceed type in order to return a value from **fquery**. When no handler handlers the condition, **fquery** proceeds normally and queries the user.

fquery *Flavor*

fquery is a simple condition built on **condition**. It is signalled by the **fquery** function when its **:signal-condition** option is **t**. The messages examine the arguments given to the **fquery** function.

Message	*Value returned*
:options	Returns the first argument to the **fquery** function.
:format-string	Returns the second argument to the **fquery** function (its format control string or prompt).
:format-args	Returns the rest of the arguments to the **fquery** function (the arguments to its format control string).

The **:choice** proceed type is provided. It has one argument, which is a value to be returned from the call to the **fquery** function.

The following example answers "yes" to every "Delete this entry?" query occurring inside **do-it** that has **:signal-condition t**:

```
(condition-bind
  ((fquery #'(lambda condition)
           (and (send condition ':proceed-type-p ':choice)
                (equal (send condition ':format-string
                             "Delete this entry?")
                (values ':choice t)))))
  (do-it))
```

2.1.11 Locking packages

package-declare and **pkg-create-package** now lock the superior package against interning. **pkg-create-package** now accepts an optional fourth argument to inhibit locking the superior. **pkg-create-package** does not affect the superior's lock when the superior already has an inferior.

si:pkg-locked *package*

si:pkg-locked is an accessor for determining whether a package is locked against interning. It returns **t** when a package is locked. You can use **setf** to clear or set the lock or use **si:with-package-lock** to bind it.

```
(si:pkg-locked (pkg-find-package 'global))
T
```

si:with-package-lock *pkg lock-value* *Macro*

si:with-package-lock binds the lock for *package* to *lock-value* around the body of the macro.

2.1.12 global and system packages locked against interning new symbols

The **global** and **system** packages are now locked against interning new symbols. Attempting to intern a new symbol in one of these packages results in an error. This change prevents the problems caused by indiscriminate loading of files into the global package. For example, two symbols that were supposed to be distinct could become the same. The correct way for a user program to put a symbol in the global or system package is with **globalize**.

A new error message warns about attempts to intern any symbols not currently in a locked package:

```
sys:foo
>>Error: Attempt to intern FOO in locked package SYSTEM
```

Using RESUME in the Debugger does permit you to intern the symbol anyhow.

2.1.13 load signals errors

load takes an optional third argument *nonexistent-ok*. In previous releases, you used this variable to make **load** return when a file could not be opened. Now, the argument means that **load** returns when **fs:file-not-found** is signalled. Other kinds of reasons for having the file not found, such as the host being down or the directory not existing, are signalled as different errors. For example, **load** now fails when the host is down even when you specified the *nonexistent-ok* argument.

2.1.14 Name changed: return-list to values

The **return-list** form to **declare** and **local-declare** has been replaced by **values**.

Old: `(declare (return-list ...))`
New: `(declare (values ...))`

See *Lisp Machine Manual*, pp. 151, 201. The old syntax continues to be supported.

2.1.15 Variable removed: fs:*always-merge-type-and-version*

fs:*always-merge-type-and-version*, announced in *System 210 Release Notes* as an experiment, has been removed from the system. Its functionality has been replaced by the new behavior of pathname merging.

2.1.16 Variable removed: fs:last-file-opened

fs:last-file-opened (*Lisp Machine Manual*, p. 381) has been removed from the system.

2.1.17 lexical-closure removed

The **lexical-closure** feature, announced in *System 74 Release Notes*, has been removed. The feature was only partially installed and was not documented. New support for lexical closures will appear with Common Lisp in a future release.

2.1.18 Changes to compiler variables

The compiler uses a set of variables and functions to keep track of which functions have been defined and which have been referenced. These are the basis for the messages "FOO was defined but never referenced" that occur during compiling.

The following variables, used for this purpose, have been reimplemented as hash tables instead of lists:

sys:file-local-declarations
compiler:functions-defined
compiler:functions-referenced

compiler:function-referenced *fspec by-whom*
This is unchanged. See the *Lisp Machine Manual*, p. 204.

compiler:function-defined *fspec*
function-defined tells the compiler that the function *fspec* has been defined (by putting it into the hash table in **compiler:functions-defined**).

compiler:file-declare *thing declaration value*
file-declare enters a declaration in the table **sys:file-local-declarations** for the remaining extent of the compilation environment.

```
(compiler:file-declare 'foo 'special t)
```

compiler:file-declaration *thing declaration*
file-declaration looks up a declaration in the table **sys:file-local-declarations**. It returns the declaration when *thing* is a declaration of type *declaration* and **nil** otherwise.

2.2 New features

2.2.1 Lambda-list keyword &key for keyword arguments

Lambda lists now have a facility that supports both positional arguments and keyword arguments. This supersedes the keyword conventions discussed in the *Lisp Machine Manual*, p. 23. This change is upward-compatible; existing compiled code does not need to be recompiled.

Keyword arguments have advantages for long lists of optional arguments where the actual calling sequences are expected to be "sparse". Using keywords, you do not have to remember the order of arguments or the default values for the arguments you are not supplying.

All positional arguments must appear before any keyword arguments in the lambda list. After **&key** in the lambda list, all the other arguments are expected to be in keyword form.

Keyword arguments are always optional, regardless of whether the lambda list contains **&optional**. Any **&optional** appearing after the first keyword argument has no effect.

&key and **&rest** are independent. They can both appear and they both use the same arguments from the argument list. The only rule is that **&rest** must appear before **&key** in the lambda list.

The **&rest** parameter is bound to a list of all the remaining arguments, both keyword symbols and values. This is just the standard meaning for **&rest**. For the keywords, the rest of the list is treated as keyword/value pairs. Each keyword parameter is bound to the value that follows it in the argument list. The following series of examples shows the main features of **&key**.

```
(defun foo (&key a b) ...)
```

a and **b** are both keyword parameters. The following example illustrates that the arguments can appear in either order.

```
(foo ':b 69 ':a '(some elements))
```

```
(defun foo (x &optional y &rest z &key a b) ...)
```

In this definition, **x** is a required positional argument. **y**, **z**, **a**, and **b** are all optional but **y** is positional, **a** and **b** are keywords, and **z** is the rest parameter. Any call with one or more arguments would be valid here. The first two satisfy the positional parameters; any others satisfy both the rest parameter and the keyword parameters. For example:

```
(foo 1 2 ':b '(6 8))
```

Argument	*Value*
x	1
y	2
z	(:b (6 8))
a	**nil**
b	(6 8)

```
(defun foo (&key a b (c working-storage-area)))
```

a, **b**, and **c** are keyword parameters. **c** has a default value supplied.

```
(defun foo (&rest z &key a b c &allow-other-keys) ...)
```

z is rest, and **a**, **b**, and **c** are keyword parameters. **&allow-other-keys** says that absolutely any keyword symbols can appear among the arguments. These symbols and the values that follow them have no effect on the keyword parameters but do become part of the value of **z**.

The rest of this section presents the detailed rules for binding keyword arguments to values.

The arguments for the keyword parameters are treated as a list of alternating keyword symbols and associated values. Each symbol is matched with the keyword parameter names; the matching keyword parameter is bound to the value which follows the symbol. All the remaining arguments are treated in this way.

The keyword symbols are compared by means of **eq**, which means they must be specified in the correct package (conventionally the keyword package). The keyword symbol for a parameter has the same print name as the parameter but resides in the keyword package regardless of what package the parameter name itself resides in. You can specify the keyword symbol explicitly in the lambda list if you need it in some package other than the keyword package or if you want the keyword to have a different name from the parameter. You do this by supplying a keyword and a variable name instead of just the variable name. The following example defines a keyword **:release** to supply a value for the variable **il:*stamp-for-release***.

```
(defun foo (&key ((:release il:*stamp-for-release*) nil)) ...)
```

Keyword parameters, like any other optional parameter, can have a default-form and a supplied-p

argument. When a keyword parameter remains unbound after all arguments have been processed, the default-form for the parameter is evaluated and the parameter is bound to its value.

&allow-other-keys is a lambda-list keyword that says to ignore any keyword symbols among the arguments that do not match any keyword parameter names. The function can access these symbols and values through a rest parameter. One common application for this is to allow a function to check only for certain keywords and then to pass its rest parameter to another function using **lexpr-funcall** with the expectation that the next function checks for the keywords that concern it. Passing unknown keywords signals an error unless **&allow-other-keys** is present.

2.2.2 New function: readline-trim

readline-trim &optional *stream eof-option options*

readline-trim trims leading and trailing white space from string input. "White space" means spaces or tabs. It takes the same arguments as the normal **readline**.

```
(readline-trim)   exciting option   RETURN
"exciting option"
(readline-trim)RETURN
""
```

The **:string-trim** and **:string-or-nil** keywords (in **prompt-and-read** and in choose-variable-values menus) use **readline-trim**.

2.2.3 New one-armed conditionals: when and unless

Two special forms have been added to make code that tests conditions more readable. These can replace compositions of **and** and **or** that implement one-armed condition testing.

when *test body...* *Macro*

The forms in *body* are evaluated when *test* returns non-null. In that case, it returns the value(s) of the last form evaluated. When *test* returns **nil**, **when** returns **nil**.

```
(when (eq 1 1) (setq a b) "foo") =>
"foo"
(when (eq 1 2) (setq a b) "foo") =>
NIL
```

When *body* is empty, **when** always returns **nil**.

unless *test body...* *Macro*

The forms in *body* are evaluated when *test* returns **nil**. It returns the value of the last form evaluated. When *test* returns something other than **nil**, **unless** returns **nil**.

```
(unless (eq 1 1) (setq a b) "foo") =>
NIL
(unless (eq 1 2) (setq a b) "foo") =>
"foo"
```

When *body* is empty, **unless** always returns **nil**.

These definitions are compatible with the current Common Lisp design.

2.2.4 Wildcard pathname mapping

Release 4.0 has a new facility for wildcard pathnames. As part of this, a new concept was introduced, that of mapping one wildcard pathname into another. In addition, three new messages were provided for **fs:pathname**.

In handling wildcard pathnames, you have to understand what it means to specify wildcards for both source and destination pathnames. For example, what does it mean to ask to copy *foo*.lisp to *bar*.lisp? In order to determine what it means, you need these two pathnames, called the source and target patterns, and a third pathname that is the starting instance. From these three ingredients, the system fashions the target pathname:

Source pattern:	`f:>fie>*old*.lisp`
Target pattern:	`vx:/usr2/fum/*older*.1`
Starting instance:	`f:>fie>--oldfoo.lisp`
Target instance:	`vx:/usr2/fum/--olderfoo.1`

A more abstract description of this terminology:

Source pattern	A pathname containing wild components.
Target pattern	A pathname containing wild components.
Source instance	A pathname that matches the source pattern.
Target instance	A pathname specified by applying the common sequences between the source and target patterns to the source instance.

Two Zmacs commands now accept pairs of wildcard file specs (see section 5.3.1, beginning on p. 88).

Copy File
Rename File

The components of the target instance are determined component-by-component for all components except the host. (The host component is always determined literally from the source and target patterns; it cannot be wild.) The mapping of pathnames is done in the native case of the target host. The source pattern and source instance are coerced to the target host via the **:new-default-pathname** message (p. 16 before the mapping takes place. When the type of the target pattern is **:wild**, it uses the canonical type for the target, regardless of the surface form for the type in the source pattern and instance.

This facility does not offer "true" wildcard mapping of directories. The semantics of wildcard mapping when hierarchical directories are involved would not be well defined. The directory component of a target pattern can be either **:wild** or a literal.

Remember!

In the Lisp Machine file system, * as the directory portion of a file spec specifies a relative pathname. You must use ** to indicate a wild directory component (see section 4.2.2, p. 75).

Here are the rules used in constructing a target instance, given the source and target patterns and a particular source instance. This set of rules is applied separately to each component in the pathname. In the mapping rules, a * character as the only contents of a component of a file spec is considered to be the same as the keyword symbol **:wild**. The rule uses the patterns from the example above.

1. If the target pattern does not contain *, copy the target pattern component literally to the target instance.
2. If the target pattern is **:wild**, copy the source component to the target literally with no further analysis. The type component is handled somewhat differently — when source and target hosts are of different system types, it uses the canonical-type mechanism to translate the type.
3. Find the positions of all * characters in the source and target patterns. Take the characters intervening between * characters as a literal value. Literal values for the name component:
   Source: old
   Target: older
4. Find each literal value from the source pattern in the source instance. Take the characters intervening between literal values as a matching value for the * from the source pattern. The matching value could be any number of characters, including zero. Matching values for the name component:
   -- and foo
5. Create the component by assembling the literal and matching values in left to right order, substituting the matching values where * appears in the target pattern. For the name component:
   --olderfoo

   When not enough matching values are available (due to too few * in the source pattern) use the null string as the matching value. When the source pattern has too many *, ignore the first extra * and everything following it.

Some examples:

Source pattern	*Source instance*	*Target pattern*	*Target instance*
*report	6802-report	*summary	6802-summary
lmfs-*	lmfs-errors	*	lmfs-errors
l*	l	l*	l
l*	lisp	l*	lisp
OLD-DIR	OLD-DIR	NEW-PLACE	NEW-PLACE
*	doc	*-extract	doc-extract
doc	doc	doc-extract	doc-extract

A set of new messages to **fs:pathname** manipulate wildcard pathnames.

:pathname-match *candidate-pathname* &optional *(match-host t)* to **fs:pathname** *Method*

:pathname-match determines whether *candidate-pathname* would satisfy the wildcard pattern of the pathname receiving the message. (The pathname receiving the message is assumed to be one that would satisfy **:wild-p**.) It compares corresponding components in the pattern pathname and *candidate-pathname*. It returns **nil** when *candidate-pathname* does not satisfy the pattern; otherwise it returns something other than **nil**.

match-host determines whether it requires the host component of the pattern to match as well. When *match-host* is **nil**, it ignores the host component. By default, it does require that the host component match.

A pattern pathname containing no wild components matches only itself.

:translate-wild-pathname *target-pattern-pathname starting-pathname* to **fs:pathname** *Method*

:translate-wild-pathname produces a new pathname based on *starting-pathname* and the analogies between the pathname receiving the message and *target-pattern-pathname*.

:translate-wild-pathname examines the correspondences between *target-pattern-pathname* and the pathname receiving the message. It then does whatever is necessary to *starting-pathname* to transform it into the target pathname.

It checks to be sure *starting-pathname* matches the pathname receiving the message and signals **ferror** if they do not match. A standard way for generating *starting-pathname* is to send **:directory-list** to the source pattern pathname to generate a set of starting pathnames.

:wild-p to **fs:pathname** *Method*

:wild-p is a predicate that determines whether the pathname is syntactically a wildcard pathname. For file spec strings, this means that the character * appears anywhere in the value of any component; for pathname objects, it means that a component is **:wild** or contains the character *.

Value	*Meaning*
nil	No component of the name is syntactically a wildcard.
not **nil**	One or more components of the name are syntactically wild. The actual value in this case is the symbol for the most significant wild component: **:device**, **:directory**, and so on.

2.2.5 New function: prompt-and-read

prompt-and-read *type format-string* &rest *format-args*

prompt-and-read prompts the user, with *format-string* and its arguments as the prompt. It uses **format** to **query-io** to produce the prompt; it reads from the **query-io** stream, calling the reading function associated with the *type* keyword. It returns whatever it reads.

This is an appropriate function to call for collecting input from the user. Its main advantages are that it does type checking on the input and takes care of redisplaying the prompt at appropriate times (for example, after the screen has been refreshed or after a notification arrives).

```
(prompt-and-read ':number "Please enter a number: ") =>
Please enter a number: 4
4
(prompt-and-read ':string "Please enter a string: ") =>
Please enter a string: 4
"4"
```

It expects to collect input of type *type*, where *type* is a keyword. It checks the type of the input and gives the user an opportunity to correct input of the wrong type. It handles the following types of input:

Option	*What terminates input*
:eval-form	Reads a Lisp form. Evaluates it and returns the first value.

:eval-form-or-end Reads a Lisp form or just END. Evaluates it and returns the first value for a form. Returns two values, **nil** and **#\end**, for END.

:expression Reads a Lisp expression. (It returns the expression without evaluating it.)

:number Reads and returns a number, terminated by SPACE or RETURN.

:pathname Reads a pathname, merging it with defaults. (See the example below for supplying the defaults.)

:string Reads a string terminated by RETURN. It returns the empty string when the string is empty.

:string-or-nil Reads a string terminated by RETURN. It trims any leading or trailing white space. It returns **nil** when the string is empty.

:string-trim Reads a string terminated by RETURN. It trims any leading or trailing white space. It returns the empty string when the string is empty.

The *type* argument can be a list in which the first argument is the type option and the rest are keyword/value pairs to serve as arguments to the reading function. In Release 4.0, only the **:pathname** type uses this kind of syntax, in order to pass in pathname defaults for merging.

```
(prompt-and-read '(:pathname :defaults ,my-defaults-alist)
                 "Enter a name (default is ~A) "
                 (fs:default-pathname my-defaults-alist))
```

Streams are permitted to have a handler for **:prompt-and-read** messages. The **prompt-and-read** function first determines whether the **query-io** stream handles the **:prompt-and-read** message. If so, it sends a **:prompt-and-read** message with its own arguments on to the stream. The stream returns several values. The first value the stream returns says whether or not it wants to handle the interaction with the user itself. It returns **nil** to indicate that it declines to handle the message, in which case the **prompt-and-read** function continues its normal action of prompting the user. When the first value is not **nil**, the **prompt-and-read** function returns the rest of the values to its caller.

2.2.6 New message to streams: :string-in

:string-in *eof string* &optional *(start 0) end* to **si:input-stream** *Method*

:string-in is a stream input operation for reading characters from an input stream into *string*, using the substring delimited with *start* and *end*. The **:string-in** message is not currently available for streams that do input from windows (for example, the keyboard); this will be changed in a future release.

As usual with strings, *start* defaults to 0 and *end* defaults to the length of the string. The difference between end and start constitutes a character count for this operation.

eof specifies stopping actions.

Value	*Meaning*
nil	Reading characters into the string stops either when it has transferred the specified character count or when it reaches end-of-file, whichever happens first. For strings with a fill pointer, it sets the fill pointer to point to the location following the last one filled by the read.
not **nil**	If the end-of-file is encountered while trying to transfer a specific number of characters, it signals **sys:end-of-file**, with the value of *eof* as the report string.

:string-in returns two values. The first value is one greater than the last location of *string* into which it stored a character. The second value is **t** if it reached end-of-file and **nil** if it did not. Using **:string-in** at the end of a file has the following effect: it returns 0 and **t** and sets the fill pointer of *string* to *start* (if *string* has a fill pointer).

For example, suppose the file my-host:>george>tiny.text contains "Here is some tiny text.".

```
(setq string (make-array 100 ':type 'art-string ':fill-pointer 0))
""

(with-open-file (stream "my-host:>george>tiny.text")
  (send stream ':string-in nil string))
23

string => "Here is some tiny text."
```

2.2.7 New facility: asynchronous characters

Programs can now interpret keyboard input *asynchronously*, that is, as soon as the character is typed, even if the program is still processing previous input. A special system process called the keyboard process calls a user-defined function as soon as the key is pressed. The main process of the program is left undisturbed. This function runs in parallel with the main program and could communicate with it.

Asynchronous character handling is available to any window that includes **tv:stream-mixin**. The window has a list that associates keyboard characters with functions. The default list contains c-ABORT, c-BREAK, c-m-ABORT, and c-m-BREAK. (See **:asynchronous-characters**, p. 31.)

The keyboard process checks each character coming in to see if it is defined as an asynchronous character for the selected window. When it is, the keyboard process calls the associated function in the context of the keyboard process.

The function that runs as a result of an asynchronous character is running in the keyboard process. It is called with two arguments, the character and **self**. It should be very short and must not do any IO. An error in one of these functions would break the keyboard process and the keyboard along with it and you would have to warm boot. To avoid any possibility of errors, you can have the function create a new process with **process-run-function** and make the new process handle the real work.

Windows with asynchronous character handling handle the following messages.

:asynchronous-character *spec-plist* (for **tv:stream-mixin**) *Init Option*

Specifies the asynchronous characters for the window. *spec-plist* is a list of specs, each of which is a list containing a character name and a function spec. The following default asynchronous characters are defined for **tv:stream-mixin**:

```
(:default-init-plist
   :asynchronous-characters '((#\c-abort kbd-asynchronous-intercept-character)
                              (#\c-m-abort kbd-asynchronous-intercept-character)
                              (#\c-break kbd-asynchronous-intercept-character)
                              (#\c-m-break kbd-asynchronous-intercept-character)))
```

:asynchronous-character-p *character* to **tv:stream-mixin** *Method*

Returns non-null when *character* is an asynchronous character for this window.

:handle-asynchronous-character *character* to **tv:stream-mixin** *Method*

Finds the function associated with *character* in the asynchronous characters list. It calls the function with two arguments, *character* and **self**. This is mainly for use by the Keyboard Process although user processes can use it also.

:add-asynchronous-character *character handler* to **tv:stream-mixin** *Method*

Defines a new asynchronous character for the window. *character* is the character to be treated asynchronously and *handler* is the function to be called (with two arguments). It checks the types of the arguments.

:remove-asynchronous-character *character* to **tv:stream-mixin** *Method*

Removes an asynchronous character from the list for the window.

2.2.8 New macros: with-open-file-case, with-open-stream-case

Two new macros provide hybrids of **with-open-stream** and **condition-case**. These support error handling for streams and files. These are similar to **with-open-file** and **with-open-stream** except that they have a set of condition clauses instead of a body. If an error is signalled, the first argument takes on the condition object as its value. Refer to *Signalling and Handling Conditions* for further information.

with-open-file-case *(variable pathname . options)* &rest *clauses* *Macro*

with-open-file-case opens a file, binding the input stream to *variable*, using the pathname and options given in the arguments. In the following example, it executes the first clause when the file is not found. When the file is found without error, it executes the second clause, which is the real reason for trying to open the file in the first place.

```
(with-open-file-case (x "f:>dla>foo.lisp" ':direction ':input)
  (fs:file-not-found (send x ':report error-output))
  (:no-error (stream-copy-until-eof x standard-output)))
```

Any errors other than **file-not-found** (for example, access violations or an unresponsive host) cause an error to be signalled normally.

with-open-stream-case *(variable stream-creation-form)* &rest *clauses* *Macro*

with-open-stream-case opens a stream and binds it to *variable*, using *stream-creation-from* to create it. It then executes whichever clause is appropriate, given the condition that resulted from the attempt to create the stream. Refer to the example shown for **with-open-file-case**.

2.2.9 New global stream: debug-io

The Debugger uses the new stream **debug-io** for all of its output and interactions. This stream is initially a synonym stream for **terminal-io**. **debug-io** replaces **eh:error-handler-io**, which is now obsolete.

The previous debugger stream, **error-output**, is now to be used only for warnings. It does not handle input operations. You can bind it to an output-only stream.

2.2.10 New stream facility for editor buffers

A new facility is available for doing I/O from streams based on Zmacs buffers.
zwei:open-editor-stream opens a stream to an editor buffer; it is analogous to **open** for files.
zwei:with-editor-stream opens a bidirectional stream to an editor buffer; it is analogous to **with-open-file** for file I/O.

2.2.10.1 New macro: zwei:with-editor-stream

zwei:with-editor-stream *(name options) body ...* *Macro*

zwei:with-editor-stream opens a bidirectional stream called *name* to a buffer, which is designated in one of the following ways:

an interval
a buffer name
a Zwei window
a pathname

It takes the same keyword options as **zwei:open-editor-stream**. See the list of options starting on p. 34.

On exit, it sends a **:force-redisplay** message to the stream, which causes the editor to do any necessary redisplay.

2.2.10.2 New function: zwei:open-editor-stream

zwei:open-editor-stream *options*

zwei:open-editor-stream is used by **zwei:with-editor-stream**. You might sometimes need to call it directly for doing operations that need not be in the scope of a "with" form (for the same reasons that you would use **open** instead of **with-open-files** for file I/O). For example, you would use this in conjunction with **with-open-stream-case** for appropriate error signalling.

It takes the same keyword options as **zwei:with-editor-stream**. See the list of options starting on p. 34.

You can send a **:force-redisplay** message at any time while the stream is open.

2.2.10.3 Keyword options

zwei:with-editor-stream and **zwei:open-editor-stream** both recognize the same set of keyword options. Some of the options are mutually exclusive and some are interdependent.

You specify where to find the text by using one of the following keywords, whichever is appropriate to the situation. The keywords appear here in priority order. When the function

options specify several of these, one from the top of the list overrides one from further down in the list, regardless of what order the keywords appear in the options list.

:interval
:buffer-name
:pathname
:window
:start

The options refer to an object called a *bp*. This is a Zwei data structure for representing a particular position in a buffer.

Option — *Values and meaning*

:buffer-name — The full name of a buffer to use for the stream.

```
(zwei:with-editor-stream
  (foo ':buffer-name (send zwei:*interval* ':name))
  ...)
```

The buffer does not need to exist (see **:create-p**). The following example creates a Zmacs buffer named temp and opens the stream **foo** to it.

```
(zwei:with-editor-stream (foo "temp")
  ...)
```

:create-p — Specifies what to do when the buffer does not exist. This applies only in conjunction with **:buffer-name** or **:pathname** with **:load-p**.

Value	*Meaning*
:ask	Queries the user before creating the buffer.
:error	Signals an error and provides proceed types for creating it or supplying an alternate.
t	Creates the buffer.
:warn	Notifies the user that a buffer is being created (the default).

:defaults — Specifies the pathname defaults against which a **:pathname** option would be merged. These are necessary in case reprompting needs to occur. The default is **nil**, meaning to use the default defaults. This option applies only in conjunction with **:pathname**.

:end — Specifies the conditions for terminating the stream (the "end of file" condition).

Value	*Meaning*
bp	Stops when this buffer bp is reached.
:end	Stops at the end of the buffer (the default). This applies only if **:start** was also a bp.
:mark	Stops when it reaches the mark. This option requires that you use the **:window** option as well.
:point	Stops when it reaches point. This option requires that you use the **:window** option as well.

:hack-fonts — Specifies how to treat font shifts in the buffer.

Value	*Meaning*
nil	Ignores font shifts (the default).
t	Provides full font support. Encodes font shifts on both input and output using epsilons, as would go to a file.

:interval — Specifies a Zwei interval to use for the stream.

:kill — Specifies what to do with the buffer before using it as a stream.

Value	*Meaning*
nil	No action (the default)
t	Deletes all the text currently in the designated part of the buffer.

:load-p Specifies whether to read the file specified by **:pathname** into the editor before using the buffer as a stream. (This is analogous to Find File in Zmacs.) This works only from within Zmacs.

Value	*Meaning*
nil	No action (the default)
t	Loads the file into the editor.

:ordered-p States whether **:start** and **:end** are guaranteed to be in forward order. The default is **nil**. This applies only when **:start** and **:end** are bps or **:point** and **:mark**.

:pathname Specifies a **pathname** to use for the stream. This can be a pathname object or any file spec that can be coerced to a pathname by **fs:parse-pathname**.

:start Specifies where to start the stream with respect to the buffer contents.

Value	*Meaning*
:append	Starts at the end of the buffer. (Same as **:end**.)
:beginning	Starts at the beginning of the buffer.
bp	Starts with this bp.
:end	Starts at the end of the buffer (the default). (Same as **:append**.)
:mark	Starts at the mark, which does not move as a result. This requires a Zmacs window.
:point	Starts at point, which does not move as a result. This requires that you use the **:window** option as well.
:region	Starts at point and ends at mark (or vice versa, depending on the ordering). This requires that you use the **:window** option as well. It ignores any **:end** in this case.

:window Specifies a Zmacs window as the stream source.

with-editor-stream does not currently interlock to prevent simultaneous access to a single buffer by multiple processes. Neither does anything else. Trying to access the same buffer with several processes simultaneously is not guaranteed to work.

2.2.11 New type: null

Lisp now recognizes a new type called **null**. The only value that has this type is **nil**. For example, when **x** is **nil**:

```
(typep nil 'null) ==> t
(setq x nil)
(typep x 'null) ==> t
```

This is compatible with the current Common Lisp design.

2.2.12 New function: let-globally-if

let-globally-if *predicate varlist* &body *body...* *Macro*

let-globally-if is like **let-globally** (*Lisp Machine Manual*, p. 17). It takes a predicate form as its first argument. It binds the variables only if *predicate* evaluates to something other than **nil**. *body* is evaluated in either case.

2.2.13 Reader macro for infix expressions

#◇ is a reader macro that turns infix expression syntax into regular Lisp code. It is intended for people who like to use traditional arithmetic expressions in Lisp code. It is not intended to be extensible or to be a full programming language. We do not intend to extend it into one.

```
(defun my-add (a b)
  #◇a+b◇)
```

The quoting character is backslash. It is necessary for including special symbols (such as -) in variable names.

! reads one Lisp expression, which can use this reader-macro inside itself.

#◇ supports the following syntax:

Delimiters — Begin the reader macro with #◇, complete it with ◇.

```
#◇a+b-c◇
```

Escape characters — Special characters in symbol names must be preceded with backslash (\). You can escape to normal Lisp in an infix expression; precede the Lisp form with exclamation point (!).

Symbols — Start symbols with a letter. They may contain digits and underscore characters. Any other characters need to be quoted with \.

Operators — It accepts the following classes of operators. Arithmetic operator precedence is like that in FORTRAN and PL/I.

Operator	*Infix*	*Lisp Equivalent*
Assignment	x : y	(setf x y)
Functions	f(x,y)	(f x y) -- also works for defstruct accessors, etc.
Array ref	a[i,j]	(aref a i j)
Unary ops	+ - not	*same*
Binary ops	+ - * / ^ = ≠ < ≤ > ≥ and or	*same*
Conditional	if p then c	(if p c)
	if p then c else a	(if p c a)
Grouping:	(a, b, c)	(progn a b c) -- even works for (1+2)/3

The following example shows matrix multiplication using an infix expression.

```
(defun matrix-multiply (a b)
  (let ((n (array-dimension-n 2 a)))
    (unless (= n (array-dimension-n 1 b))
      (ferror "Matrices ~S and ~S do not have compatible dimensions") a b)
    (let ((d1 (array-dimension-n 1 a))
          (d2 (array-dimension-n 2 b)))
      (let ((c #◊ make\-array(list(d1, d2), !':type, art\-float)◊ ))
        (dotimes (i d1)
          (dotimes (j d2)
            #◊ c[i,j] : !(loop for k below n sum #◊ a[i,k]*b[k,j] ◊)◊))
        c))))
```

The line containing the infix expression could also have been written like this:

```
(let ((sum 0))
  (dotimes (k n) #◊ sum:sum+a[i,k]*b[k,j] ◊)
  #◊ c[i,j]:sum ◊)
```

2.2.14 New arguments to :insert-char, :delete-char, :insert-line, and :delete-line

The window system messages **:insert-char, :delete-char, :insert-line,** and **:delete-line** now accept a second optional argument. It specifies the units as either **:character** or **:pixel**. The default is **:character,** which gives the coordinates in character widths and line heights for the window. (The meaning of character widths and line heights is the same as for **:set-cursorpos**. See *Introduction to Using the Window System.*)

2.2.15 New property functions for flavors

si:flavor-default-init-putprop *flavor value property*

si:flavor-default-init-putprop is just like **putprop** except that its first argument is either a flavor structure or the name of a flavor. It puts the property on the default init plist of the specified flavor.

si:flavor-default-init-get *flavor property*

si:flavor-default-init-get is just like **get** except that its first argument is either a flavor structure or the name of a flavor. It retrieves the property from the default init plist of the specified flavor. You can use **setf**:

```
(setf (si:flavor-default-init-get f p) x)
```

si:flavor-default-init-remprop *flavor property*

si:flavor-default-init-remprop is just like **remprop** except that its first argument is either a flavor structure or the name of a flavor. It removes the property from the default init plist of the specified flavor.

2.2.16 New keyword: :fill-pointer

:fill-pointer A keyword for **make-array**. It causes **make-array** to give the array a fill pointer and initializes it to the value following the keyword. Use this instead of **:leader-length** or **:leader-list** when you are using the leader only for a fill pointer. This keyword is compatible with the current Common Lisp design, which has no array leaders.

2.2.17 New variable: compiler:compiler-verbose

compiler:compiler-verbose *Variable*

The compiler displays a message (using **standard-output**) each time it starts compiling a function when the value is **t**. The default value is **nil**.

2.2.18 New special form: undefun-method

undefun-method *function-spec* *Special Form*

undefun-method undoes the effect of **defun-method** in the same way that **undefmethod** undoes the effect of **defmethod**. (See *Lisp Machine Manual*, p. 297.) This is a special form, not a function, so *function-spec* is not evaluated.

2.2.19 New defflavor option: :abstract-flavor

:abstract-flavor is a new option to **defflavor**. It declares that the flavor exists only to define a protocol; it is not intended to be instantiated by itself. Instead, it is intended to have more specialized flavors mixed in before being instantiated.

Trying to instantiate an abstract flavor signals an error.

:abstract-flavor is an advanced feature that affects paging. It decreases paging and usage of virtual memory by allowing abstract flavors to have combined methods. Normally, only instantiated flavors get combined methods, which are little Lisp functions that are automatically built and compiled by the flavor system to call all of the methods that are being combined to make the effective method. Sometimes many different instantiated flavors use the same combination of methods as each other. If this is the case, and the abstract flavor's combined methods are the same ones that are needed by the instantiated flavors, then all instantiated flavors can simply share the combined methods of the abstract flavor instead of having to each make their own. This sharing improves performance because it reduces the working set.

compile-flavor-methods is permitted on an abstract flavor. It is useful for combined methods that most specializations of that flavor would be able to share. This was one reason for this change. Without the **:abstract-flavor** declaration, **compile-flavor-methods** warned you about the flavor not being one that could be instantiated.

2.2.20 New functions: fs:enable-capabilities, fs:disable-capabilities

fs:enable-capabilities *host* &rest *capabilities*

fs:enable-capabilities is a host-independent mechanism for enabling the privileges required on a remote host for certain kinds of file or directory manipulation (for example, creating a new directory).

```
(fs:enable-capabilities "comet" "bypass")
```

The capabilities argument consists of strings that name the specific capabilities to enable. The capability names are host-dependent. The defaults are those appropriate to a particular type of host. For example,

Host	*Defaults*
TOPS-20	WHEEL and OPERATOR
VAX/VMS	SYSPRV

It returns an alist with elements of the following form:

(*capability-name* . *status*)

Each element of the alist is a string naming a capability that you have the right to enable. *status* is **t** or **nil** to indicate whether it is currently enabled.

When the file job needs to open a second connection to a host that has capabilities enabled, the second connection also has capabilities enabled. When an enabled file job loses its connection and reconnects, the capabilities are reenabled.

The editor command Enable Host Capabilities provides an alternate interface to this function (see section 5.2.1.5, p. 86).

fs:disable-capabilities *host* &rest *capabilities*

fs:disable-capabilities is a host-independent mechanism for disabling the privileges required on a remote host for certain kinds of file or directory manipulation (for example, creating a new directory).

The capabilities argument consists of symbols that name the specific capabilities to disable. The capability names are host-dependent. The defaults are those appropriate to a particular type of host. It returns an alist with elements of the following form:

(*capability-name* . *status*)

capability-name is a string; *status* is **t** or **nil**.

The editor command Disable Host Capabilities provides an alternate interface to this function (see section 5.2.1.5, p. 86).

2.2.21 New message: :draw-circular-arc

:draw-circular-arc *center-x center-y radius start-theta end-theta* &optional *(alu char-aluf)* to **tv:graphics-mixin** *Method*

Draws a circular arc for the circle centered at *center-x*, *center-y* with radius *radius*. It draws the part of the circle swept counterclockwise from the starting angle to the finishing angle. The angles are assumed to be in radians and are reduced mod 2pi before drawing. For example, drawing from pi/4 to -pi/4 draws a "C". The same "C" appears when you draw from pi/4 to 7pi/4.

For **tv:alu-xor**, the behavior with respect to points that would fall on the same pixel is not defined.

2.2.22 New message: :draw-closed-curve

:draw-closed-curve *x-array y-array* &optional *end (alu char-aluf)* to **tv:graphics-mixin** *Method*

:draw-closed-curve draws a sequence of connected line segments, using the points in *x-array* and *y-array* as the *x* and *y* coordinates for the end-points of the lines. It ensures that each particular point is drawn only once, which is necessary for producing a

connected line with **tv:alu-xor**. It plots the points in the arrays until *end* elements or until it encounters **nil** in either of the arrays. The default for *end* is the length of *x-array*. *alu* specifies how the pixels being drawn combine with those already there. The default *alu* is **char-aluf** for the window. It plots the points in the arrays until *end* elements or until it encounters **nil** in either of the arrays.

:draw-closed-curve is the same as **:draw-curve** (see *Introduction to Using the Window System*) except that it closes the figure by joining the first and last points.

2.2.23 New message: :draw-dashed-line

:draw-dashed-line *from-x from-y to-x to-y* &optional *(alu char-aluf) (dash-spacing 20.) space-literally-p (offset 0) dash-length* to **tv:graphics-mixin** *Method*

:draw-dashed-line draws a dashed line along the line lying between two points. All the dashes are the same length; all the spaces between the dashes are the same length. (The spaces, however, need not be the same length as the dashes). The spacing and lengths of the dashes are controlled by separate arguments.

alu	Controls how the pixels being drawn combine with pixels already in the window. The default is the **char-aluf** for the window.
dash-spacing	Specifies the distance from the beginning of one dash to the beginning of the next dash. It is expressed in pixels. The default is 20. (The spacing between dashes is *dash-spacing* minus *dash-length*.) This specifies the "frequency" of the line.
space-literally-p	Controls what happens when the distance between the points, given the specified spacings, would not produce a full-size dash connected to the endpoint. The default value, **nil**, allows the size of *dash-spacing* to be adjusted slightly so that the dashes are all of equal size and both endpoints look the same, as far as dash length goes. In this case, the dash-length is always exactly half of the dash-spacing; any values for *offset* and *dash-length* are ignored. The value **t** means to use *dash-spacing* exactly, with no adjustment. The endpoint might or might not have a dash connected to it, depending on the exact distances involved.
offset	Specifies a distance (in pixels) from the starting point (*from-x*, *from-y*) for the beginning of the first dash. This lets you control the "phase" of the dashed line.
dash-length	Specifies the length of the line segments, in pixels. It must be less than *dash-spacing*. This lets you control the "duty cycle" of the line.

You can make complex dashing by using **:draw-dashed-line** many times with *space-literally-p* as **t**. For example:

```
(progn
 (send terminal-io ':draw-dashed-line 0 0 200. 200. tv:alu-ior 25. t 0 10.)
 (send terminal-io ':draw-dashed-line 0 0 200. 200. tv:alu-ior 25. t 15. 5.))
```

This gives you alternating long and short dashes. Because the **nil** value for *space-literally-p* changes the spacing, this technique does not work well when *space-literally-p* is **nil**.

2.2.24 New message: :draw-string

:draw-string *string from-x from-y* &optional *(toward-x (1+ from-x)) (toward-y from-y) (stretch-p nil) (font (sheet-current-font self)) (alu char-aluf)* to **graphics-mixin** *Method*

:draw-string draws a character string between two points. It returns the location of the last character printed.

The string can contain either normal printing characters or **art-fat-string** characters with font change codes. The left baseline point of each character lies on the line between the two points defined by *from-x*, *from-y* and *toward-x*, *toward-y*. It uses the baseline rather than the upper-left corner to ensure that strings with mixed fonts line up properly.

The string is always written from left to right, starting at the leftmost point, regardless of whether that is the first point or the second point. When the string is longer than the line between the points, the full string appears anyhow.

toward-x, *toward-y*
: Controls the direction in which printing takes place. The default values specify ordinary horizontal output.

```
(send (tv:window-under-mouse) ':draw-string
      "hi there" 600 50)
```

stretch-p
: Controls the spacing of the characters. When it is **nil** (the default), the characters appear literally, with no change to the spacing. Otherwise, the distance between the characters is adjusted so that the string starts and ends as close to the two points as possible.

font
: Specifies the font to use. The default is the current font for the window.

alu
: Controls how the pixels being drawn combine with pixels already in the window. The default is the **char-aluf** for the window.

This message is useful for placing text at absolute screen positions (as opposed to treating the window as a stream), for labelling graphs, or for putting text into pictures.

2.2.25 New function: sys:%slide

sys:%slide *initial-half-wavelength delta-half-wavelength delta-time duration*

sys:%slide is a function for producing sound using the keyboard audio. It is like **sys:%beep** but changes the wavelength by *delta-half-wavelength* every *delta-time* microseconds. It can produce ascending or descending tones, depending on the sign of *delta-half-wavelength*.

```
(sys:%slide 0 50 10000 1000000) =>
4950
(sys:%slide 5000 -50 10000 1000000) =>
50
```

It returns the final half-wavelength, for use in composite effects.

All of the arguments must be small fixnums. Only *delta-half-wavelength* can be negative. When this function is running, it uses the entire processor.

2.2.26 New function: signum

signum *value*

signum is a function for determining the sign of its argument.

```
(signum -2.5)  =>  -1.0
(signum 3.9)  =>  1.0
(signum 0)  =>  0
(signum 59)  =>  1
```

The definition is compatible with the current Common Lisp design.

2.2.27 New function: string-capitalize-words

string-capitalize-words *string* &optional *(copy-p t)*

Transforms *string* by changing hyphens to spaces and capitalizing each word.

```
(string-capitalize-words "Lisp-listener") => "Lisp Listener"
(string-capitalize-words "LISP-LISTENER")  => "Lisp Listener"
(string-capitalize-words "lisp--listener") => "Lisp  Listener"
(string-capitalize-words "symbol-processor-3") => "Symbol Processor 3"
```

copy-p indicates whether to return a copy of the string argument or to modify the argument itself. The default, **t**, returns a copy.

2.2.28 New function: parse-number

parse-number *string* &optional *(from 0) (to nil) (radix nil) (fail-if-not-whole-string nil)*

parse-number takes a string and "reads" a number from it. It returns two values: the number found (or **nil**) and the character position of the next unparsed character in the string. It returns **nil** when the first character that it looks at cannot be part of a number. The function currently does not handle anything but integers.
(**read-from-string** is a more general function that uses the Lisp Reader;
prompt-and-read, p. 29 reads a number from the keyboard.)

```
(parse-number "123    ")  =>  123 3
(parse-number "   123")  =>  NIL 0
(parse-number "-123")  => -123 4
(parse-number "25.3")  =>  25 2
(parse-number "$$$123" 3 4)  =>  1 4
(parse-number "123$$$" 0 nil nil nil) => 123 3
(parse-number "123$$$" 0 nil nil t) => NIL 0
```

Four optional arguments:

from — The character position in the string to start parsing. The default is the first one, position 0.

to — The character position past the last one to consider. The default, **nil,** means the end of the string.

radix — The radix to read the string in. The default, **nil,** means base 10.

fail-if-not-whole-string — The default is **nil.** **nil** means to read up to the first character that is not a digit and stop there, returning the result of the parse so far. **t** means to stop at the first non-digit and to return **nil** and 0 length if that is not the end of the string.

2.2.29 New dump function: sys:dump-forms-to-file

sys:dump-forms-to-file *file forms-list* &optional *attribute-list*

sys:dump-forms-to-file writes data to a file in binary form. *forms-list* is a list of Lisp forms, each of which is dumped in sequence. It dumps the forms, not their results. The forms are evaluated when you load the file.

For example, suppose **a** is a variable bound to any Lisp object, such as a list or array. The following example creates a compiled code file that recreates the variable **a** with the same value:

```
(sys:dump-forms-to-file "f:>foo>aval"
        (list '(setq a ,a)))
```

This function is a generalization of the functions **compiler:fasd-symbol-value**, **compiler:fasd-font**, and **compiler:fasd-file-symbol-properties** (*Lisp Machine Manual*, p. 207). (**compiler:fasd-file-symbol-properties** has been removed from the system.) For the purposes of understanding what this does, you can consider that it is the same as the following:

```
(defun sys:dump-forms-to-file (file forms)
   (with-open-file (s file ':direction ':output)
      (dolist (f forms)
        (print f s))))
```

The real definition writes a binary file so it will load faster. It can also dump arrays, which you cannot write to a Lisp source file.

attribute-list supplies an optional attribute list for the resulting compiled code file. It has basically the same result when loading the binary file as the file attribute list does for **qc-file**. Its most important application is for controlling the package that the file is loaded into.

```
(sys:dump-forms-to-file "foo" forms-list '(:package "user"))
```

This function makes **compiler:fasd-symbol-value** obsolete.

Old:	`(compiler:fasd-symbol-value)`
New:	`(sys:dump-forms-to-file '((setq ,sym ',val)) ...)`

2.2.30 New special forms: with-stack-list and with-stack-list*

The new special forms **with-stack-list** and **with-stack-list*** are subprimitives. They cons lists on the control stack so that when you are finished, the lists are popped off without leaving any garbage. This is essentially giving you access to the mechanism that **&rest** arguments use. Because these are on the control stack, you cannot return the lists that they cons, use **rplaca** or **rplacd** with them, or place references to them in permanent data structures.

with-stack-list *(variable* &rest *list-elements) body* *Special Form*

with-stack-list is used to bind a variable to a list and evaluate some forms in the context of that binding. It is like **let** (in that it binds a variable) except that it conses the list on the stack.

```
(with-stack-list (var element1 element2...elementn)
  body)
is like
(let ((var (list element1 element2...elementn)))
  body)
```

with-stack-list* *(variable &rest list-elements) body* *Special Form*

with-stack-list* uses **list*** instead of **list**. See *Lisp Machine Manual*, p. 57, for a description of **list***.

```
(with-stack-list* (var element1 element2...elementn)
  body)
is like
(let ((var (list* element1 element2...elementn)))
  body)
```

2.2.31 New functions: location-makunbound and location-boundp

location-makunbound and **location-boundp** are versions of **makunbound** and **boundp** that can be used on any cell in the Lisp Machine. They take a locative pointer to designate the cell rather than a symbol. (**makunbound** is restricted to use with symbols.) The following two calls are equivalent:

```
(location-boundp (locf a))
(variable-boundp a)
```

The following two calls are also equivalent. When **a** is a special variable, they are the same as the two calls in the preceding example too.

```
(location-boundp (value-cell-location 'a))
(boundp 'a)
```

2.2.32 New optional arguments to string-search

string-search and **string-reverse-search** both accept two new optional arguments. These allow you to specify a substring of the search key. That is, instead of using the whole search key, you can use part of it. The default is to use the whole search key so this change is completely upward compatible. The new argument lists are as follows:

string-search *key string* &optional *(from 0) to (key-start 0) key-end*

string-reverse-search *key string* &optional *(from 0) to (key-start 0) key-end*

2.2.33 New type of method combination: :append

:append is a new type of simple method combination (see the *Lisp Machine Manual*, p. 307). All the component methods are called as arguments to **append**. It expects each of the methods to return a list; the final result is the result of appending all these lists. No typed methods are allowed.

2.2.34 New type of method combination: :nconc

:nconc is a new type of simple method combination (see the *Lisp Machine Manual*, p. 307). All the component methods are called as arguments to **nconc**. It expects each of the methods to return a list; the final result is the result of concatenating these lists. No typed methods are allowed.

2.2.35 New type of method combination: :case

:case is a new type of method combination that takes a subsidiary message name. It dispatches on this message name just as the original message name caused a primary dispatch. This facility is used in the condition handling system. (See *Signalling and Handling Conditions.*)

```
(defmethod (sys:subscript-out-of-bounds :case :proceed :new-subscript)
           (&optional (sub (prompt-and-read ':number
                                            "Subscript to use instead: ")))
  "Supply a different subscript"
  (values ':new-subscript sub))

(send obj ':proceed ':new-subscript new-sub)
```

2.2.36 evalhook accepts optional apply hook argument

The **evalhook** function now accepts an optional third argument, an *apply hook.* (See the explanation of **evalhook** in the *Lisp Machine Manual*, p. 466.)

Normally after **eval** has evaluated the arguments to a function, it calls the function. If an apply hook exists, however, **eval** calls the hook with two arguments: the function and its list of arguments. The values returned by the hook constitute the values for the form. The hook could use **apply** on its arguments to do what **eval** would have done normally. This hook is active for special forms as well as for real functions.

Whenever either an evalhook or applyhook is called, both hooks are bound off. The evalhook itself can be **nil** if only an applyhook is needed.

The applyhook catches only **apply** operations done by **eval.** It does not catch **apply** called in other parts of the interpreter or **apply** or **funcall** operations done by other functions such as **mapcar**. In general, such uses of **apply** can be dealt with by intercepting the call to **mapcar,** using the applyhook, and substituting a different first argument.

The argument list is like an &rest argument: it might be stack-allocated but is not guaranteed to be. Hence you cannot perform side-effects on it and you cannot store it in any place that does not have the same dynamic extent as the call to the "applyhook".

2.2.37 New function: record-source-file-name

record-source-file-name *function-spec* &optional *(type* **'defun***) no-query*

record-source-file-name associates the definition of a function with its source files, so that tools such as Edit Definition (m-.) can find the source file of a function. It also detects when two different files both try to define the same function, and warns the user.

record-source-file-name is called automatically by **defun, defmacro, defstruct,**

defflavor, and other such defining special forms. Normally you do not invoke it explicitly. If you have your own defining macro, however, that does not expand into one of the above, then you can make its expansion include a **record-source-file-name** form.

function-spec The function spec for the entity being defined.

type The type of entity being defined, with **defun** as the default. *type* can be any symbol, typically the name of the corresponding special form for defining the entity. Some standard examples:

defun
defvar
defflavor
defstruct

Both macros and substs are subsumed under the type defun, because you cannot have a function named **x** in one file and a macro named **x** in another file.

no-query Controls queries about redefinitions. **t** means to suppress queries about redefining. The default value of *no-query* depends on the value of **inhibit-fdefine-warnings**. When **inhibit-fdefine-warnings** is **t**, *no-query* is **t**; otherwise it is **nil**. Regardless of the value for *no-query*, queries are suppressed when the definition is happening in a patch file.

You cannot specify the source file name with this function. The function is always associated with the pathname for the file being loaded (**fdefine-file-pathname**).

2.2.38 New function: store-conditional

store-conditional replaces the function **%store-conditional** which was the basic locking primitive (*Lisp Machine Manual*, p. 178). It accepts only locative arguments. **%store-conditional** continues to exist for compatibility as a macro that expands into **store-conditional**.

store-conditional behaves like **%p-store-contents** in that it leaves the cdr code and the flag bit of the location that is being stored into undisturbed.

2.2.39 Compiler scheme for keeping track of macros

The compiler now records, as part of its debugging-info property, which top-level macros were expanded in the process of compiling it. This information is used by **who-calls** and similar functions. Thus you can now use **who-calls** for macros. **who-calls** can also find callers of open-coded functions, such as substitutable functions. Functions compiled in earlier versions of the system have not recorded this information; hence **who-calls** will not be able to find them until those sources have been recompiled. (See also **sys:function-parent**, p. 104.)

2.2.40 New features in defstruct

defstruct has three new options: **:print**, **:predicate**, and **:copier**. The **:print** option allows you to control the printed representation of a structure. The **:predicate** option causes **defstruct** to generate a predicate that recognizes instances of the structure. The **:copier** option causes **defstruct** to generate a function for copying instances of the structure.

2.2.40.1 :print option

The **:print** option gives you implementation-independent control over the printed representation of a structure.

```
(defstruct (foo :named
                (:print "#<Foo ~S ~S>" (foo-a foo) (foo-b foo)))
  foo-a
  foo-b)
```

The **:print** option takes a format string and its arguments. The arguments are evaluated in an environment in which the name symbol for the structure is bound to the structure instance being printed.

People used to use a **named-structure-invoke** handler to define **:print** handlers. This is no longer necessary; the **:print** option does it for you.

2.2.40.2 :predicate option

The **:predicate** option causes **defstruct** to generate a predicate that recognizes instances of the structure. The first example defines a single-argument function, **foo-p**, that returns **t** only for instances of structure **foo**. The second example defines a function called **is-it-a-foo?**.

```
(defstruct (foo :named :predicate)
  foo-a
  foo-b)
(defstruct (foo :named (:predicate is-it-a-foo?))
  foo-a
  foo-b)
```

The **:predicate** option has one optional argument, the name for the function being generated. The default name for the generated function is formed by appending "-p" to the structure name.

The **:predicate** option is restricted to work only for named types.

2.2.40.3 :copier option

The **:copier** option causes **defstruct** to generate a function for copying instances of the structure.

```
(defstruct (foo (:type list) :copier)
  foo-a
  foo-b)
```

This example would generate a function named copy-foo, with a definition approximately like this:

```
(defun copy-foo (x)
  (list (car x) (cadr x)))
```

2.2.40.4 General improvements to defstruct

A number of general improvements have been installed.

- You can use alterant macros on structures whose accessors require additional arguments. Put the additional arguments before the list of slots and values, in the same order as required by the accessors.
- The **:displace** option has been removed. Macros defined by **defstruct** never displace.
- The **:make-array** option to defstruct lets you control the initialization of arrays created by **defstruct** as instances of structures. **make-array** initializes the array before the constructor code does. Therefore any initial value supplied via the new **:initial-value** keyword for **make-array** is overwritten in any slots where you gave **defstruct** an explicit initialization.

2.2.41 New options for defstruct-define-type

defstruct-define-type accepts two new options: **:predicate** and **:copier**. The **:predicate** option specifies how to construct a **:predicate** option for **defstruct**. The **:copier** option specifies how copy a particular type of structure.

2.2.41.1 :predicate option

The **:predicate** option specifies how to construct a **:predicate** option for **defstruct**.

```
(:predicate (description name)
            `(defun ,name (x)
               (and (frobbozp x)
                    (eq (frobbozref x 0)
                        ',(defstruct-description-name)))))
```

The syntax for the option follows.

```
(:predicate (description name)
            body)
```

The variable *description* is bound to the defstruct-description structure maintained for the structure we are to generate a predicate for. The variable *name* is bound to the symbol that is to be defined as a predicate. *body* is a piece of code that is evaluated to return the defining form for the predicate.

2.2.41.2 :copier option

The **:copier** option specifies how to copy a particular type of structure for situations when it is necessary to provide a copying function other than the one that **defstruct** would generate.

```
(:copier (description name)
  `(fset-carefully ',name 'copy-frobboz))
```

The syntax for the option follows.

```
(:copier (description name)
         body)
```

description is bound to an instance of the defstruct-description structure, *name* is bound to the symbol to be defined, and *body* is some code to evaluate to get the defining form.

2.3 Improvements

2.3.1 Prompt provided by yes-or-no-p and y-or-n-p

The two "yes or no" functions for querying the user now supply a prompt that indicates which form of answer (single letter or full word plus RETURN) is required. This prompt is appended to any message that you supply with the function. (See **fquery**, p. 22.)

```
(y-or-n-p "More? ") =>
More? (Y or N) Yes.

(yes-or-no-p "Detonate terminal? ") =>
Detonate terminal? (Yes or No) no
```

Remove any redundant "(Y or N)" style prompting from your functions.

2.3.2 Data types returned by numeric functions

Transcendental functions, like **sin**, that return numeric values now return a value that is the same type as the type of the argument. That is, for small floating number arguments, they return small floating number values.

2.3.3 ~$ format directive for floating point

~$ is a previously undocumented **format** directive for floating point values.

The format for using it follows:

*~rdig,ldig,field,padchar***$**

It expects a flonum argument. The modifiers for **~$** are all optional.

rdig	The number of digits after the decimal point. The default is 2.
ldig	The minimum number of digits before the decimal point. The default is 1. It pads on the left with leading zeros.
field	The full width of the field to print in. The default is the number of characters in the output. The field is padded to the left with *padchar* if necessary.
padchar	The character for padding the field if the field is wider than the number. The default is **#\space**.
:	The sign character is to be at the beginning of the field, before the padding, rather than just to the left of the number.
@	The number must always appear signed.

```
(format t "~&Pi is ~$" (atan 0 -1))  =>
Pi is 3.14
(format t "~&Pi is ~8$" (atan 0 -1))  =>
Pi is 3.14159265
(format t "~&Pi is ~8,2@:$" (atan 0 -1))  =>
Pi is +03.14159265
(format t "~&Pi is ~8,2,20$" (atan 0 -1))  =>
Pi is          03.14159265
(format t "~&Pi is ~8,,20,'x@$" (atan 0 -1))  =>
Pi is xxxxxxxxx+3.14159265
```

It uses free format (**~@A**) for very large values of the argument.

2.3.4 Warning about redefining functions

Lisp Listeners now warn you when you evaluate a form that tries to redefine a function that has already been defined.

```
WARNING: Function FOO being redefined
```

It prompts you about whether to continue with the redefinition. This is designed to prevent you from damaging your Lisp environment accidentally by redefining a system function. The same check and warning are now issued when you evaluate a form or buffer in the editor in both file buffers and nonfile buffers.

2.3.5 Stack growth

When it is necessary to grow the regular stack, it now grows by a reasonably large ratio, set by **dbg:pdl-grow-ratio,** instead of just growing by a little bit as it did before.

2.3.6 Window system: :notice protocol changed

The **:notice** methods are combined with **or** method-combination. **:notice :error** returns one of the following:

nil	Go on.
t	Do not call methods of more basic flavors; otherwise okay.
a string	This window cannot be used for debugger interaction; the string explains why not.

2.3.7 Changes in flavor instantiation

A small change was made to initializing flavor instances. Now **:default-init-plist** entries that initialize instance variables are not added to the init-plist, whereas formerly all **:default-init-plist** items that were not overridden were added to the init-plist. Entries in the original init-plist supplied to **make-instance** that initialize instance variable remain visible to **:init-methods.** This change was made as part of a performance improvement in instantiating a flavor.

One minor change resulted for flavors that put things on the default init plist (where **a** is an instance variable):

```
(:default-init-plist :a 'foo)
```

The following construct in a **:init** method now returns **nil** rather than **foo.** Use the value of **a** instead.

```
(get (locf init-plist) ':a)
```

2.3.8 declare recognized in blocks

The compiler now recognizes **declare** as the first forms in the bodies of the following:

let	**let***
do	**do***
do-named	**do*-named**
prog	**prog***
lambda	

Formerly, **declare** was recognized only as the first forms in the body of a function. This means that you can now have **special** declarations that are local to any of these blocks.

2.3.9 unwind-protect restrictions removed

In the past, **unwind-protect** has had some restrictions associated with **go, return,** and multiple value returns. These have been removed. **unwind-protect** now works correctly in all cases.

2.3.10 defsubst now compiled

A *subst* is an open-coded function created by **defsubst**. **defsubst** now creates compiled functions, where they were previously only interpreted. Thus a subst is now faster when invoked as a function. As a consequence of changes to the implementation of subst, it is now possible to get new compiler warnings when compiling files containing **defsubst**. For example, the following **defsubst** would not have previously gotten a warning, even though **x** is free in **add-with-x**.

```
(defsubst add-with-x (y) (+ x y))
```

The current implementation would issue a warning because substs are now implemented with compiled code objects. (This example would still work if expanded in an environment which lexically contained **x**.)

2.3.11 Warning about obsolete make-array form

The compiler now warns about uses of the obsolete form of **make-array** that does not use keywords. (See the *Lisp Machine Manual*, p. 113.) It continues to accept the obsolete form, however.

2.3.12 Compiler no longer expands macros in a different area

The compiler no longer expands macros in the fasd temporary area. It now puts them into the area designated by **default-cons-area** at the time of compilation. This eliminates a source of very difficult-to-find bugs that occurred only with sophisticated macros.

Use the garbage collector if your macros cons a lot.

3. Utilities

3.1 Incompatible changes

3.1.1 Change in naming for compiled-code files

Compiled-code files now have the type QBIN instead of QFASL. You can just rename your QFASL files to QBIN with no problem; they do not need to be recompiled. The load functions no longer recognize QFASL types. The actual file types for compiled-code files are host-dependent of course:

Host type	*File type for compiled files*
ITS	QBIN
Lisp Machine	qbin
Multics	qbin
TENEX	QBIN
TOPS-20	QBIN
UNIX	qb
VAX/VMS	QBN

This change is not backward-compatible; files compiled in Release 4.0 cannot be loaded into a Lisp Machine running System 210 or some other vendor's software. The compiler generates some new instructions. This is why the file type was changed from QFASL to QBIN.

3.1.2 Password prompting change

The previous means of prompting for a password or user id caused some confusion because it did not echo the user id. This has been changed. It expects a password and does not echo the typein. When you need to change the user id as well as give a password, start the typein with ALTMODE. Type the new user id, followed by RETURN. Then type the password. The user id is now echoed as you type it; the password does not echo. The dialog would look like the following:

```
Default user name for host K2 is George
Password (or Altmode to change user id): ALTMODE [New user id]
Enter user name for host K2: JWalker RETURN
Password (Altmode to change user id):
```

3.1.3 Package changes

Due to internal reorganization, some packages and systems have been renamed, some have been removed, and new ones have appeared. The following table explains the details.

System 210	Release 4.0
qfasl-rel	*gone*
canon	lgp
lbp	*gone*
as68000	*gone*
eh	dbg (debugger)
none	mailer
none	print (can be loaded manually)

3.1.4 Debugger changes

See also the new Debugger features in sections 3.2.8, p. 63, 3.2.9, p. 63, and 3.2.10, p. 63; improvements in section 3.3.2, p. 69.

The Debugger now uses multiple arrow prompts to signal recursive invocations of the Debugger.

The Debugger provides a list of ways to stop or proceed on each error, with each option bound to one of the SUPER- keys. For example, s-A might be equivalent to RESUME and s-B to ABORT. The old Debugger commands ABORT, c-ABORT, and c-Z still operate, using **sys:abort** instead of ***catch**. See details of Debugger command keys in *Signalling and Handling Conditions.*

c-S now searches from the current frame rather than the top frame. It offers a default when you enter a blank line. This makes it possible to search through several calls to a given function. The search command in the window debugger uses the same default.

Backtraces (for example, c-B) now start at the current frame rather than the top frame. A numeric argument specifies how many frames to print.

m-< now gets you to the frame that signalled the error, rather than the highest possible frame. Use c-P to go up through **signal**, handlers, and so forth, until you get to the call to the Debugger itself. At that point, c-P reports that you are at the top of the stack; this is where m-< used to put you.

When a function is being traced, the Debugger now shows you that fact and displays the traced function as a single frame. This makes it easier to use the Debugger in the presence of traced functions. c-m-N and c-m-P continue to show the internals of what tracing did.

3.1.5 eh function renamed to dbg

dbg is the new name for the function **eh**. See the *Lisp Machine Manual*, p. 451. The function has changed slightly.

- **dbg** runs in the process itself instead of stopping the process as **eh** used to.
  ```
  (dbg p)
  ```
- With no argument, it enters the Debugger as if an error had occurred for the current process. It is not an error; in particular, **errset** and **catch-error** do not handle it. You can include this form in program source code as a means of entering the Debugger.
  ```
  (dbg)
  ```
  This is useful for breakpoints and causes a special compiler warning.
- With an argument of **t** (rather than a process, window, or stack group), it finds a process that has sent an error notification.
- **dbg** sets up a restart handler for c-Z and ABORT that exits from the **dbg** function back to the original process. The message for this restart handler is "Allow process to continue".
- You can now use c-T, c-R, c-m-R, and other similar Debugger commands when you enter the Debugger via **dbg**.

Suppose you are running in process *X* and you use **dbg** on some process *Y*. Process *Y* is forced into the Debugger, no matter what it is doing. Technically, it is "interrupted", similar to how

c-BREAK, c-ABORT and c-m-BREAK work. Process *Y* starts running the Debugger, using the stream **debug-io**. **debug-io** gets the same stream as was bound to **terminal-io** in Process *X*. At this time, Process *X* waits in a state called DBG until Process *Y* leaves the Debugger and so does not contend for the stream.

3.1.6 Debugger function names changed

The following Debugger functions and variables have been renamed. (See *System 210 Release Notes*, p. 18.)

Old	*New*
eh-arg	**dbg:arg**
eh-loc	**dbg:loc**
eh-fun	**dbg:fun**
eh-val	**dbg:val**
eh-sg	no longer available
eh-frame	**dbg:*frame***

3.1.7 New value for rubout-handler

The variable **rubout-handler** indicates the status of rubout handling for input to any process. (See *Lisp Machine Manual*, p. 362.) It now has one of three values instead of two:

Value	*Meaning*
nil	The process is outside the rubout handler.
read	The process is inside the **:rubout-handler** method.
tyi	The process is inside the editing portion of the **:tyi** method.

This change affects the internals of code that handles RUBOUT; it probably does not affect any user programs.

3.1.8 Rubout handler change for LINE

LINE has been changed from an ordinary input character to an editing character. It now has the same operation as RETURN when you are typing to something like a Lisp listener.

3.1.9 CALL key function removed

The CALL key no longer arrests the current process and calls a Lisp Listener. It now has no function. TERMINAL CALL still calls the cold-load stream.

3.1.10 Finger error changes

The functions **chaos:finger** and **chaos:whois** no longer return a string when there is an error. Instead, they signal an appropriate condition. If you want to handle this, the right error flavor to handle is **sys:network-error**. (See also Signalling and handling conditions, p. 17.)

3.1.11 Conditionalizing on sites

In previous systems, you could determine the site name at read time with #+ or the **status** function. This has been replaced with a more useful site name checking facility that works at run time. This makes it possible to move compiled code that contains site dependencies from one place to another and have it still run.

You still use the following to find the list of status keywords for features available at a site.

```
(status features)
```

The following list summarizes the changes to the status features.

- **site-name** is now a global variable, not a status feature. Its value is the site name keyword declared in **defsite**. #+ and #- no longer work for site names.
- One item indicates which Lisp Machine hardware is running. **:cadr** indicates an LM-2 or an MIT CADR. **3600** is another possibility. You would then conditionalize your code with things like `#+cadr` or `#-3600`.
- One item indicates which kind of Lisp Machine software is running. **:symbolics** indicates Symbolics software; for example, use `#-symbolics`.

3.1.12 Patch directories have new generation retention count

Patch file directories are now automatically created with a generation retention count of 2, if they are part of a file system that supports generation retention count.

3.2 New features

3.2.1 Garbage collector improvements

Release 4.0 contains a number of improvements to the garbage collector, associated with turning it on before you run out of address space, and not turning it on after it is too late to do any good. These improvements include queries when you turn on garbage collection too late, notifications that you had better turn it on soon if you intend to, and automatic shut-off when there might not be enough space left to run it. See section 7.2.1, p. 106, for a general definition of garbage collection in the Lisp Machine and strategies for using it.

The garbage collector is stable and reliable. Please report to Symbolics any problems you encounter that you believe to be associated with garbage collection.

3.2.1.1 Controlling when garbage collection happens

gc-immediately

gc-immediately does nonincremental garbage collection, taking less space and less total time than an incremental gc, but running continuously in the process calling it, until the garbage collection is complete. The main advantage of this compared to incremental gc is that it requires less free space and hence can succeed where an incremental gc would fail because virtual memory was too full.

You should call this rather than **si:full-gc** (unless you are compressing a band). The

difference is that **gc-immediately** does not lock out other processes, does not run various **full-gc** initializations, and does not affect the static areas.

Suppose garbage collection has already started, that the flip has occurred but not all good data have been copied out of oldspace. **gc-immediately** then copies the rest of the good data but does not flip again.

gc-status

gc-status prints statistics about the garbage collector. It prints different information depending on whether the scavenger is running or finished and how full virtual memory is.

```
Dynamic (new+copy) space 4,181,235.  Old space 0.  Static space 2,590,587.
Free space 8,778,513.  Committed guess 8,685,807, putting you 169,438 words past the point
where an incremental garbage collection risks running out of space.
There are 4,335,138 words available before (GC-IMMEDIATELY) might run out of space.
(GC-IMMEDIATELY) takes roughly one hour.
There are 8,778,513 words available if you elect not to garbage collect.
Garbage collector process state: Nonexistent
Scavenging during cons: Off, Idle scavenging: Off
GC Flip Ratio: 1, GC Reclaim Immediately: Off
GC messages (controlled by SI:GC-REPORT-STREAM) are sent as notifications.
The GC generation count is 2.
```

si:gc-report-stream *Variable*

si:gc-report-stream specifies where to put output messages from the garbage collector.

Value	*Meaning*
t	Notifies you (default)
nil	Discards the output
stream	Sends output to the stream

si:gc-area-reclaim-report *Variable*

si:gc-area-reclaim-report controls reporting on the areas reclaimed.

Value	*Meaning*
nil	Does not report anything (default)
t	Reports space reclaimed for each area

si:gc-warning-threshold *Variable*

si:gc-warning-threshold controls the "switch" for turning on GC warnings. When the storage manager notices that the amount of free space remaining before it would be too late to garbage collect has reached *limit*, it notifies you that you need to turn on the GC. The default value is 1000000.

si:gc-warning-ratio *Variable*

si:gc-warning-ratio controls how often you see warnings from the GC (after the warnings threshold has been passed) before running out of space. Basically, it is as if you had reset the warning threshold to be *fraction* times the previous warning threshold. For example, the default ratio is 0.75. With the default values for **si:gc-warning-threshold** and **si:gc-warning-ratio**, you would see GC warnings with 1000000, 750000, 562500, 421875, and so on words remaining.

si:gc-flip-ratio *Variable*

si:gc-flip-ratio contains a number that approximately expresses the maximum fraction of dynamic space that you expect to contain good data (as opposed to garbage). This applies only after **gc-on**. It specifies when a flip takes place. When this number times the amount of committed free space (committed to being used by the GC) is greater than the amount of free space, then a flip occurs. The default value is 1.

The number can be less than 1. This would cause the GC to wait longer before flipping at the risk of exhausting virtual memory if a larger fraction of dynamic space contains good objects than you expected.

See **si:gc-flip-minimum-ratio** also for a discussion of finer control over the onset of garbage collection.

si:gc-flip-minimum-ratio *Variable*

si:gc-flip-minimum-ratio contains a number that specifies when to turn the GC off because memory is too full to allow copying anything. The default value is **nil**, which specifies that this ratio has the same value as **si:gc-flip-ratio**. Otherwise it should be a number less than **si:gc-flip-ratio**.

Putting 0.25 in **si:gc-flip-minimum-ratio** and 0.5 in **si:gc-flip-ratio** means that you believe that less than 0.25 of the dynamic-space objects consed are good data and will need to be copied by the garbage collection. In spite of this, you want to flip when there is enough space to copy 0.5 (half) of the objects. Thus one controls how often the GC flips; the other controls when it should get desperate. This is most useful if you turn on **si:gc-reclaim-immediately-if-necessary**, to make it do something useful when it is desperate. Even without that it is useful if you would rather risk doing a garbage collection when there might not be enough memory left in preference to turning the GC off. For example, when the machine is operating unattended and turning off the GC would be guaranteed to make it exhaust memory.

Choosing good values for this variable is a matter of guesswork and experience with the particular application.

si:gc-reclaim-immediately *Variable*

si:gc-reclaim-immediately affects the "await scavenge" state of the garbage collector. When the value is **nil**, (the default), the garbage collector is not affected. When the value is not **nil**, then **gc-reclaim-oldspace** occurs as soon as a flip has occurred. This essentially removes the real-time aspect of the garbage collector, making it more like an ordinary copying garbage collector.

si:gc-reclaim-immediately-if-necessary *Variable*

si:gc-reclaim-immediately-if-necessary controls whether the GC starts nonincremental garbage collection or shuts down. This variable is irrelevant when **si:gc-reclaim-immediately** is set because then the GC always reclaims immediately, even if it does not need to.

The variable controls what happens when not enough free space remains to copy everything. When the value is **nil** (the default), it notifies you and turns itself off. For other values, it tries nonincremental garbage collection and shuts itself off only when it determines that nonincremental garbage collection is not going to work.

It is possible for so little space to remain that even a nonincremental garbage collection would exhaust virtual memory. The decisions about what would exhaust virtual memory depend on your prediction of the fraction of dynamic space that contains real (nongarbage) objects (This is the value of **si:gc-flip-minimum-ratio**.)

si:set-scavenger-ws *ws-size*

si:set-scavenger-ws sets the working set size for the garbage collector. The default size is 64K. The size is expressed in units of pages, where **sys:page-size** is 256 (1/4 K).

```
(si:set-scavenger-ws 256.)
80128.
```

This is the amount of memory that the Scavenger (the asynchronous portion of the garbage collector) is permitted to use for its paging. The remainder of memory is reserved for your program and the system.

si:gc-flip-inhibit-time-until-warning *time* *Variable*

si:gc-flip-inhibit-time-until-warning sets the reasonable time window for flipping. If flipping does not occur successfully during *time*, the GC notifies you about the problem. *time* is expressed in 60ths of a second. The default is 10 seconds. Flipping cannot occur when some program is running in an **si:inhibit-gc-flips** special form.

si:inhibit-gc-flips *Macro*

si:inhibit-gc-flips prevents the GC from flipping within the body of the macro.

3.2.1.2 Changes to si:full-gc

si:full-gc no longer takes an optional argument for reclaiming all of the static areas (*System 210 Release Notes*, p. 34). The resulting loss of locality had seriously degraded paging performance.

si:full-gc now accepts a keyword argument, **:system-release**. This feature converts existing parts of certain areas (notably working-storage-area) from dynamic to static areas. Thus objects in those areas when **full-gc** occurs become permanent and are never garbage collected. (Of course any future objects created in those areas are dynamic and are subject to garbage collection.) This effectively makes more user free space because fewer data need to be copied on each garbage collection pass.

```
(si:full-gc ':system-release t)
```

While the initializations for **si:full-gc** are being run, the variable **si:*full-gc-for-system-release*** is bound to the value of the **:system-release** keyword.

Currently **:system-release** does the following work:

- Combines equal symbol print names.
- Converts parts of working-storage-area and some regions of other areas to be static.

3.2.2 Release versions: si:get-release-version

Releases have numbers and status, just as systems do. Symbolics staff assign the release numbers, using the file SYS: SYS; RELEASE LISP. This file is distributed with new software releases and loaded by **load-patches**.

si:get-release-version

si:get-release-version returns three values, the release numbers and the status of the current world load:

Major version number
Patch version number
Status of the world load as a keyword symbol:
:experimental
:released
:obsolete
:broken
nil (when status cannot be determined)

For example:

```
(si:get-release-version) => 4 0 :released
```

3.2.3 Two speeds for cold boot

The Lisp Machine offers two styles of cold boot. One is faster than the other. The conservative cold boot process involves copying the entire world load into the virtual memory partition. The fast boot process copies very little of the world load before completing and then pages the information into virtual memory from the band as need for it is encountered. The fast process is now the default.

disk-restore has an optional second argument that specifies which style of cold boot to use. (See *System 210 Release Notes*, section 4.4.) The default for the second argument has been changed from **t** (for conservative cold boot) to **nil** (for fast cold boot). Disk saving after the fast boot is currently very slow; use the slow boot if you intend to use **disk-save**.

3.2.4 Recommended procedure for copying and saving bands

si:transmit-band, si:receive-band, and **si:copy-disk-partition** must copy into a target band that is not active. Do not try to receive a band into the one that the current environment is paging out of. Do not transmit into a band that someone is using. (It warns you if you try.)

The following steps constitute a full safe procedure for saving virtual memory with **disk-save**.

1. Cold boot. Slow cold boot is recommended; otherwise saving the band will be very slow.
2. Use **si:disable-services**, see section 8.2.8, p. 116. This prevents things like Converse messages from arriving and corrupting the image for the new band.
3. Log in without loading your init file (so that other users of the band do not use your init file).
4. Build the environment you want to save, loading your own system and its patches, and whatever is needed.
5. Use **disk-save**.

In some cases, you might have to use **disk-save** into the band that you are running from. This is not recommended unless absolutely necessary. When you do so, first set the current band to some other band, just in case some problem occurs and you need to cold boot.

3.2.5 Changes to global package

The following symbols have been added to or moved to the global package:

argument-typecase
boundp-in-instance
catch-error-restart
catch-error-restart-if
condition
condition-bind-default
condition-bind-default-if
condition-bind-if
condition-call
condition-call-if
condition-case
condition-case-if
copyf
dbg
debug-io
describe-system (previously **si:describe-system**)
error-restart-loop
fsignal
gc-immediately
gc-status
ignore-errors
instancep
let-globally-if
listf
location-boundp
location-makunbound
login-forms
make-condition
prompt-and-read
qreply
readline-trim
record-source-file-name
signal-proceed-case
signum
string-capitalize-words
trace-conditions
undefun-method
unless
when
with-open-file-case
with-open-stream-case
with-stack-list
with-stack-list*

The following symbols have been removed from the global package.

dplt-print-file
dtp-stack-closure
eh-arg
eh-frame
eh-fun
eh-loc
eh-sg
eh-val
lexical-closure
print-txt-file

3.2.6 Hardcopy commands available

All menu items, commands, and functions that refer to printing and hard copy are now supported. In order for them to actually work, your site must have a properly connected printing device (see section 8.2.2, p. 112). Some of the commands now available:

[Hardcopy] in the system menu
Hardcopy Buffer (m-X) in the editor
Hardcopy File (m-X) in the editor
Hardcopy Region (m-X) in the editor
[Move / Hardcopy] in Zmail
Dired Print File, P subcommand of Dired in the editor
[*filename* / Hardcopy] in the file system editor
TERMINAL Q, screen copy command

Some sites have more than one printer. One of the printers has been specified as the site default hardcopy device. You can change the default in your init file to specify the printer that is most convenient for you. In the system menu, [Hardcopy] allows you to specify a different printer name; the printer name is mouse sensitive.

You can specify personal default fonts for each device in your init file.
si:*hardcopy-default-fonts* is an association list where each element specifies a device and a set of keyword/value pairs designating the font. The keywords are **:default-font** and **:header-font** as in section 8.2.2, p. 112. For example:

```
(login-forms
  (setq si:*hardcopy-default-fonts*
        '(("Itasca" :default-font ("Fix" "B" 18.)))))
```

si:set-default-hardcopy-device *name*
si:set-default-hardcopy-device specifies the printer to be used for all of the hardcopy commands except the screen copy command. *name* is a string specifying the device name. (The valid list of device names appears in the **:hardcopy-devices** option in the site file.)

si:set-screen-hardcopy-device *name*
si:set-screen-hardcopy-device specifies the printer to be used for screen copies (by the TERMINAL Q command). *name* is a string specifying the device name. (The valid list of device names appears in the **:hardcopy-devices** option in the site file.)

3.2.7 New FED command copies a font

FED can now copy an existing font as the first step of making a new font. Use [(mouse-M) Edit Font]. It is no longer necessary to copy the characters one a time.

3.2.8 Debugger c-M command sends mail

c-M in the Debugger is a new command for sending bug-mail that contains information from several stack frames. This uses the **bug** function. It places you in a mail sending context, initialized with the error message and a partial stack trace. By default it supplies 3 stack frames, starting with the one that signalled the error. You can supply a different number of stack frames by using the command with a numeric argument.

You are expected to supply context information explaining what you were doing when the problem occurred. The stack trace by itself is not adequate information for debugging.

3.2.9 New Debugger command: c-m-V

c-m-V sets * to the *n*th returned value, where *n* is its numeric argument. The default value for the argument is 0. It is meaningful only when you are using trap-on-exit and looking at a frame that is about to return.

3.2.10 New Debugger option: dbg:*defer-package-dwim*

When this is **nil** (the default), the Debugger searches over all packages to find any lookalike symbols, when errors concerning unbound variables occur.

When the option is not **nil**, it gives you control over whether or not to search for look-alike symbols.

3.2.11 New overprinting command: NETWORK O

In their normal mode of operation, TELNET and SUPDUP overprint the first character with the second when a remote system sends two characters to be output in the same position. A new command on NETWORK O causes SUPDUP and TELNET to first erase a character position that it is about to write into. The command toggles the overprinting facility.

In SUPDUP, the overprinting toggle must be set to the position you want before you make a SUPDUP connection; you cannot change it once a connection has been established. NETWORK HELP shows the position of the toggle (**t** or **nil**). The default is **t**, indicating overprinting.

3.2.12 New special form: login-forms

login-forms *body ...* *Special Form*

login-forms is a new special form for wrapping around a set of forms in your init file. It evaluates the forms and arranges for them to be undone when you log out. It is intended to replace **login-setq** and the other login forms.

login-forms always evaluates the forms, even when it does not know how to undo them. For forms that it cannot undo, it prints a warning message.

In the following example, **login-forms** arranges for **foo** either to become unbound or to get its old value and for **bar** either to become undefined or to get its old function definition. It would warn you about **quux** being impossible to undo.

```
(login-forms
  (setq foo 3)
  (defun bar (x y) (+ x y))
  (quux 3))
```

You can create functions to undo forms that **login-forms** does not recognize. To undo a given form, you put a property on the symbol that is the car of the form to undo. For example, to create a function to undo **quux**:

```
(defun (:property quux :undo-function) (form)
  `(undo-quux ,(cadr form)))
```

The value returned by an undo function is a form to be evaluated at logout time.

3.2.13 Addendum to band compressing procedure

A compressed band runs more slowly initially than a normal band. This is because any hash tables that depend on virtual addresses (see section 2.1.1.4, p. 6) are rehashed the first time they are referenced.

You can get around this and make compressed bands that run at normal speed by adding one additional step to the compressing procedure. After you have used **si:compress-band** to compress the band, boot the new band. Then use **disk-save**, which takes care of all the rehashing that is necessary. The procedure for compressing and saving a band was described in *System 210 Release Notes*.

3.2.14 New flexibility in make-system

make-system has a new feature for finding how out to make a system that has not been defined already. When the system it is looking for has not been defined already or been set up with **si:set-system-source-file**, it looks for system definition information in a file with the following name:

sys: site; *system-name* system

That file should contain **si:set-system-source-file**. More information appears in *Software Installation Guide*.

3.2.15 Field patches

In previous releases, trying to patch one of the distributed systems made it difficult to install a new release. Release 4.0 has a new facility for declaring which sites can patch a system and helping to monitor versions to ensure that no changes are lost.

defsystem has a new option **:maintaining-sites** to specify the list of sites that maintain the system. Currently this declaration is meaningful only for patchable systems. For example:

```
(defsystem dla-file-system
  ...
  (:maintaining-sites :mit)
  ...)
```

The default for **:maintaining-sites** when it is undeclared is the usually local site. When you attempt to distribute a system with an undeclared maintaining site, you are warned and urged to supply a maintaining site. When you save a band for distribution, all patchable systems with undeclared **:maintaining-sites** are automatically declared to be maintained at the distributing site.

When you attempt to patch a system that is not maintained at your site, you would see a warning like the following:

```
System DLA-FILE-SYSTEM is not normally maintained at this site.  Patching it
here may result in version skews and make it difficult for your site to
receive subsequent software updates.
Are you sure you really sure you want to patch it? (Yes or No)
```

3.2.16 Local site systems: :site-system option

Many sites have software packages or patches which should be loaded into all bands at the site. Sites can now declare to have such a patchable system. The system software attempts to provide more assistance in maintaining this system. The site system name can be declared using the **:site-system** option to **defsite**. The system should be declared in a file with the following kind of name:

```
sys: site; site-name system
```

If the site system is not loaded into a band, then a warning is printed in the herald to that effect.

M-X Add Patch and related commands provide the site system as the default system if no other locally maintained systems are appropriate.

3.2.17 New keyword to load-patches: :norelease

:norelease is a new keyword for **load-patches**. It inhibits the loading of the SYS: SYS; RELEASE file (explained in *Software Installation Guide*).

It is useful only when you do not specify particular system names to **load-patches**. Using an explicit system name implies **:norelease**.

Argument	*Meaning*
a system name	Never loads SYS: SYS; RELEASE
:norelease	Never loads SYS: SYS; RELEASE
neither	Always loads SYS: SYS; RELEASE

3.2.18 New functions: tv:add-escape-key, tv:add-system-key

tv:add-escape-key and **tv:add-system-key** are two new functions for defining escape and system keys. In previous releases, you used the variables **tv:*escape-keys*** and **tv:*system-keys*** for this purpose. (See *Introduction to Using the Window System*, pp. 37, 38.) The new functions replace the use of those variables, which should no longer be modified directly. The new functions ensure that items being added are sorted correctly in the documentation display.

tv:add-escape-key *(char function documentation* &rest *options)*

> **tv:add-escape-key** adds *char* to the list of keys that can follow the TERMINAL key. The rest of the arguments are explained in *Introduction to Using the Window System*, p. 37.

tv:add-system-key *(char flavor name* &optional *(create-p* t*))*

tv:add-system-key adds *char* to the list of keys that can follow the SYSTEM key. The rest of the arguments are explained in *Introduction to Using the Window System*, p. 38.

3.2.19 New variable: tv:*screen-hardcopy-announcement*

tv:*screen-hardcopy-announcement* *Variable*

tv:*screen-hardcopy-announcement* controls how TERMINAL Q notifies you. Some kind of notification is necessary after a screen hardcopy request so that you know when the information for the copy has been collected and it is thus safe to change the screen. The default value is **:beep**, which is consistent with System 210. The other possible value is **:flash**, which complements the screen when bit collection begins and sets it back to normal video when it is complete.

```
(login-forms
  (setq tv:*screen-hardcopy-announcement* ':flash))
```

3.2.20 New function: copyf

copyf *from-path to-path* &key *(characters ':default) (byte-size nil) (copy-creation-date t) (copy-author nil) (report-stream nil)*

copyf is a function for copying one file to another. Copy File in the editor uses this function.

from-path and *to-path* are the source and destination pathnames, which can be file specs. **from-path** must refer to a unique file; it cannot contain any wild components. **copyf** first attempts to open *from-path*. When that has happened successfully, it parses *to-path* and merges it (using **fs:merge-pathnames**) against the truename of *from-path* and version of **:newest**. This has the following result for version numbers.

to-path can contain wild components, which are eliminated after merging the defaults by means of **:translate-wild-pathname** (see p. 29).

Source	*Target*	*Result*
>foo>a.b.newest	>bar>	Retains the version number
>foo>a.b.newest	>bar>x	Uses version number for file w.b

copyf and **renamef** treat version number defaulting in the same way.

By default, **copyf** copies the creation date of the file but not the author. (These defaults are not optimal but have low overhead, due to a limitation of the file protocol.)

characters It decides whether this is a binary or character transfer according to the canonical type of *from-path*. You do not need to supply this argument for standard file types. Its possible values:

- **:default** For types that are not known canonical types, it opens *from-path* in **:default** mode. In that case, the server for the file system containing *from-path* makes the character-or-binary decision.
- **t** Specifies that the transfer must be in character mode.
- **nil** Specifies that the transfer must be binary mode (in this case, you must supply *byte-size*).

byte-size Specifies the byte size with which both files will be opened for binary transfers. You must supply *byte-size* when *characters* is **nil**. It determines the byte size from the file type for *from-path*. When *from-path* is a binary file with a known canonical type, it determines the byte size from the **:binary-file-byte-size** property of the type. When the file does not have a known type, it requests the byte size for *from-path* from the file server. When the server for the file system containing *from-path* cannot supply the byte size, it assumes that the byte size is 16.

report-stream When *report-stream* is **nil** (the default), the copying takes place with no messages. Otherwise, the value must be a stream for reporting the start and successful completion of the copying. The completion message contains the truename of *to-path*.

3.2.21 New function: listf

listf *path* &optional *(output-stream standard-output)*

listf is a function for displaying an abbreviated directory listing. The default for name, type, and version of *path* is **:wild**.

```
(listf "f:>jwalker>mit-220")
```

The format of the listing varies with the operating system.

3.2.22 New function: chaos:print-lgp-queue

chaos:print-lgp-queue

chaos:print-lgp-queue displays the queue of requests for each host that has a Chaosnet print server running. This function is specific to the Symbolics LGP-1 printer. It does not report the queues to other kinds of printers.

```
LGP queue for host Afghan:

For Quinsigamond: Request of 1/02/83 22:49:37 for F:>jek>serial-
io.lisp.11 for Carl W. Hoffman at SCRC-SPANIEL
For Quinsigamond: Request of 1/02/83 22:50:33 for
F:>Network>network.lisp.35 for Carl W. Hoffman at SCRC-SPANIEL
For Quinsigamond: Request of 1/02/83 22:51:28 for
F:>Network>serial.lisp.11 for Carl W. Hoffman at SCRC-SPANIEL
LGP queue for host POINTER:

The queue is empty.
```

3.2.23 New variable: chaos:finger-location

chaos:finger-location *Variable*

chaos:finger-location sets the location reported by the finger functions. Its value should be a string to print as the location part of a finger display. When this variable is **nil**, (the default), the system uses the value **si:local-finger-location**, which is set automatically by remote file servers. When the variable has a string value, it overrides the value in **si:local-finger-location**.

3.2.24 New functions: chaos:finger-local-lispms, chaos:finger-all-lispms

At sites whose host tables are completely internal and do not include Lisp Machines at other sites, these two functions have the same result.

chaos:finger-local-lispms
Displays a list of who is using each of the Lisp Machines at the current site. (It uses **si:machine-location-alist**.) This function replaces the function **chaos:finger-all-lms** from previous releases.

chaos:finger-all-lispms
Displays a list of who is using each of the Lisp Machines in the host table. (It uses **si:host-alist**.)

3.2.25 New function: qreply

qreply &optional *string*
Sends a reply to the Converse message received most recently. You can supply a string as the text of the message or omit it and let **qreply** prompt for it. It returns a string of the form "*user@host*", indicating the recipient of the message.

3.2.26 New function: tv:key-test

tv:key-test allows you to check that your keyboard and mouse hardware are functioning correctly. It displays a keyboard image and a mouse image. The mouse image tracks the mouse when mouse tracking is functioning correctly. Holding down a key or button causes the corresponding key or button on the screen to go into inverse video. The END key returns. This function is not loaded as part of the world load but is available:

```
(load "sys:window;keytest")
(tv:key-test)
```

3.2.27 New meter: sys:%count-extra-pdl-ovs

sys:%count-extra-pdl-ovs is a meter that increments each time the extra pdl area is flushed. Use **read-meter** to read it (see *Lisp Machine Manual*, p. 188).

3.3 Improvements

3.3.1 Loading patches catches network errors

In previous releases, people experienced problems with having **load-patches** in their init files in situations where the sys host was not available. Now **load-patches** is protected by a **catch-error-restart** for network errors. This means that your init file can continue executing the rest of its forms even though it cannot load patches.

3.3.2 Debugger messages have new formats

When you first enter the Debugger, it no longer displays a short backtrace of functions. The c-B and c-L commands still give backtrace information. In place of the previous backtrace, you now see a list of proceed and restart commands.

You can request that backtrace information appear when you enter the Debugger. Set the new variable **dbg:*show-backtrace*** in your init file.

Values	*Meaning*
nil	The Debugger startup message does not include any backtrace information. **nil** is the default.
t	The Debugger startup message includes a 3-element backtrace, as in System 210 and earlier systems.

The c-B command in the Debugger gives a backtrace of the call stack. It now displays the functions so that function names do not wrap across line boundaries, to make the display more readable.

3.3.3 Change to filename defaulting in renamef

renamef uses *from-path* and *to-path* as source and target pathnames, which can be file specs. It first attempts to open *from-path.* When that has happened successfully, it parses *to-path* and merges it (using **fs:merge-pathnames**) against the truename of *from-path* and version of **:newest**. This has the following impact on the version number of the target.

Source	*Target*	*Result*
>foo>a.b.newest	>bar>	Retains the version number
>foo>a.b.newest	>bar>x	Uses version number for file w.b

renamef and **copyf** treat version number defaulting in the same way. This change to **renamef** affects Rename File in the editor.

3.3.4 Status line changes

The states that appear in the status line have been revised to be both more specific and more informative. Many of the changes occurred to the names of the lock states. For example, the Zmail background process uses "Background Lock".

3.3.5 print-herald shows memory available

print-herald now includes a line that reports on the physical and virtual memory available on the machine. It reports much the same information as the function **room**.

3.3.6 Appearance of windows, borders, and labels

All system windows, even ones that are the full screen size, have borders and labels. The system windows that were most affected are Lisp Listeners and Zmacs frames. The flavor called **tv:full-screen-hack-mixin** has been removed from the standard window flavors.

Typeout windows now have a border by default. The border moves down the screen just below

the last line in the typeout window. This change makes typeout windows more distinguishable from whatever is on the rest of the screen. This change is cosmetic only; it does not require changes to any programs.

tv:*typeout-window-border-enable* *Variable*

tv:*typeout-window-border-enable* controls the border. The default value is **t**, which enables the border. **nil** inhibits the border.

As part of the general cosmetic changes, returning to Lisp Top Level now produces a message like the following instead of just a * prompt.

```
Back to Lisp Top Level in Lisp Listener 1
```

3.3.7 Window label format

Automatically generated window names now appear in mixed cases with spaces separating the words. It uses the function **string-capitalize-words** (section 2.2.27).

3.3.8 Contact names for Chaosnet connections

The contact names for Chaosnet connections are now retained in the connection data structures. The accessor function is **chaos:contact-name**.

3.3.9 Peek changes

3.3.9.1 Peek handles ABORT

Peek now handles ABORT more consistently with other programs. It no longer puts you in Help mode when you abort out of a breakpoint. ABORT at Peek's top level is now ignored, as it is in other programs. (It used to bury the peek window.)

3.3.9.2 Peek updates on SPACE

SPACE now commands Peek to update its display immediately instead of waiting for its next time interval.

3.3.9.3 Peek displays Chaos connection names

Peek now displays the contact names for Chaosnet connections. In the Peek display, the connection state field is mouse-sensitive. Clicking on the state allows you to close the connection from Peek. Closing a connection this way requires subsequent mouse confirmation.

3.3.9.4 Peek file system command

[File System] in Peek no longer displays entries for hosts that have no host units to find. Only "interesting" hosts and their host units are displayed. Resetting a host unit now requires mouse confirmation.

3.3.9.5 Peek server command

[Server] in Peek now shows more information for file server data connections and their associated processes. File server processes have more meaningful names, which include the name of the host to which they are connected.

3.3.10 TERMINAL HOLD-OUTPUT changed

TERMINAL HOLD-OUTPUT is now more useful in cases where the status line shows the states Sheet Lock or Hold Output. Now it usually manages either to expose the required window or to enter the Debugger with the right process. It is more informative about what the problem is and engages you in a dialog about how to resolve it.

3.3.11 Lisp (Edit) removed

Previous versions of the system had a Create menu item named Lisp (Edit). This was a Lisp Listener with editing capability. This has been removed from the system menu pending redesign. Lisp Listeners themselves now have a variety of editing capabilities. (See section 3.3.12, p. 71.)

Although it is not supported, Lisp (Edit) is still available as part of the system for those who care to experiment. The following code restores the function to the Create menu.

```
(push '("lisp (edit)" :value zwei:editor-top-level
        :documentation
        "A read-eval-print loop in separate process with editing capabilities.")
      tv:default-window-types-item-list)
```

3.3.12 Rubout handler improvements

The previously experimental editing capability is now on by default. (See *System 210 Release Notes*, p. 28, concerning **tv:rh-on**.) Use **tv:rh-off** to disable this editing. Use c-HELP at top level to see the editor-compatible commands that are available. This editing capability is present in any process that uses **read, readline,** or **prompt-and-read**. Some improvements:

- c-sh-V (like Describe Variable At Point in the editor) has been added to complement c-sh-A.
- HELP and c-HELP give help in the same style that the Debugger does. That is, HELP displays a brief summary and c-HELP displays a detailed help message.
- The kill ring and input rings are now emptied when you log out or save a band.

In the future, this editing capability will be expanded and made completely compatible with the editor.

3.3.13 Arresting most processes: TERMINAL c-A

Some changes were made to arresting processes. These are to allow experts to intervene in a runaway machine for purposes of debugging.

- TERMINAL c-A arrests all processes except the one shown in the status line and critical system processes, like the keyboard and mouse processes.
- TERMINAL - c-A resumes all processes arrested by TERMINAL c-A.
- When a process arrested by TERMINAL c-A shows in the status line, TERMINAL - A resumes just that process. (Remember TERMINAL 3 W for rotating through processes.)

Arresting is a severe action to take. For example, when the processes that run the Chaosnet are

arrested, your network connections are lost when you leave them stopped for more than 60 seconds. When a process that is holding a lock is arrested, it holds the lock forever, possibly interfering with subsequent normal work.

4. File System

This section applies only to the Lisp Machine file system, not to other file systems such as UNIX and TOPS-20. This material does apply equally to local and remote Lisp Machine file systems.

The Lisp Machine File System has been revised extensively internally to use the new error signalling system. Most of these changes are not visible to end-users, except as far as the diagnostic messages are more complete and informative.

The file system software has been reorganized internally for future support of multiple partitions and drives. Some of this change is visible in menu items and messages that now mention partitions. No other user-visible changes from the multiple partition redesign have been installed.

4.1 Incompatible changes

4.1.1 Files being created are invisible to most operations

In previous versions, files that were being created were available as soon as they had been opened for writing. This could cause problems for multiprocess applications where one process would try to use a file that another was still writing.

In Release 4.0, when a file version is being created, it is marked with the property **:open-for-writing**. This property is removed when the file is successfully closed. While the file has this property, it is invisible to normal directory operations and to attempts to open or list it. Directory list operations that specify **:deleted** can see the file. Files in this state have the "open for writing" property when you use View Property in the file system editor. Files left in this state by crashes have to be removed manually by deleting and expunging.

For example, suppose versions 3, 4, and 5 exist, but 5 is open in this state. An attempt to read **:newest** would get version 4; an attempt to write **:newest** would create version 6.

4.1.2 Wrong byte size error

It is not possible to open a file with the "wrong" byte size. If you are going to supply a byte size when you open a file, it must be the byte size that it was written with. (By default, the Lisp Machine file system always opens a file using the same byte size as it was written with.) Using the wrong byte size has always been an error; the file system now diagnoses and signals the error correctly.

4.1.3 Directory components have changed

The format of directory components in Lisp Machine file system pathnames has changed. This should not affect any of your programs; programs that depended on the format of directory components were in error. They should have been using the relevant ":string-for" messages instead. (See also section 2.1.6.3 on p. 15.)

4.1.4 File names cannot contain *

In previous releases, names of files stored in the Lisp Machine file system could contain the character *. That is now longer possible; file names can no longer contain *. This restriction is necessary because * is now used consistently to indicate wildcards in pathnames. Quoting conventions to relax this restriction will be implemented in a future release.

You can no longer access files whose names contain * as a character. A special function allows you to rename any file or directories whose names contain *.

lmfs:rename-local-file-tool *from-path to-path*

lmfs:rename-local-file-tool renames a file in which * appears in one of the pathname components. This function works locally only; you must run it on the machine in whose file system the file is stored. That is, it does not rename a file across the network.

from-path and *to-path* must be pathnames or strings coercible to pathnames. *from-path* is parsed against a default on the local host. *to-path* is parsed against *from-path* as the default. The version number for *to-path* is inherited from the file being renamed. Any version number appearing in *to-path* is ignored.

```
(lmfs:rename-local-file-tool ">AUser>*secret-stuff*" "-secret-stuff-")
(lmfs:rename-local-file-tool ">*special*.directory.1" "-special-")
```

4.2 New features

4.2.1 Relative pathnames

The Lisp Machine file system now supports relative pathnames. The syntax and semantics are identical to their usage in Multics. A relative pathname is relative to whatever defaults it is being merged with, for example, the displayed defaults in the Zmacs minibuffer. Pathname default merging follows these rules:

- A name with no leading punctuation adds to the end of the directory portion of the current default.
- A name with one or more leading < replaces directory name levels from the right, one directory level for each <.

Examples:

Default: `>sys>lmfs>new>xst.lisp`
Typein: `test>xst.lisp`
Merged: `>sys>lmfs>new>test>xst.lisp`

Default: `>sys>lmfs>new>xst.lisp`
Typein: `<test>thing.lisp`
Merged: `>sys>lmfs>test>thing.lisp`

Default: `>sys>lmfs>new>xst.lisp`
Typein: `<<test>`
Merged: `>sys>test>xst.lisp`

Default: >sys>lmfs>new>xst.lisp
Typein: <xst.lisp
Merged: >sys>lmfs>xst.lisp

Default: >sys>lmfs>new>xst.lisp
Typein: <<abel>baker>foo.lisp
Merged: >sys>abel>baker>foo.lisp

4.2.2 Wildcards extended

The Lisp Machine file system now supports arbitrary wildcards in pathnames. The wildcard character is *. In a pathname, * means "zero or more characters". You cannot use two consecutive asterisks except as a wildcard directory. You cannot create a directory or file with * in the name. The following examples are now valid as wildcard pathnames:

```
"f:>jwalker>*foo*.lisp"
"local:>*foo>notes.text"
"f:**>*.lisp
```

Due to the implementation of relative pathnames, *>foo now means "all files named foo in any directory one level below the merge default". The syntax "**" was chosen for a directory component of **:wild**. Therefore, the following command lists all files with type lisp anywhere at any level on the host *f*.

```
(fs:directory-list "f:**>*.lisp")
```

** is a complete specification of the directory component of a pathname. It has no meaning except at the beginning of a pathname.

The file system now does more checking for * in invalid contexts.

4.2.3 New file type for indicating directories

The file type "directory" can now be used to specify a directory. This applies for various file operations, such as View File Properties. You could still create files (rather than directories) that have "directory" as their type component. The file system takes care of knowing which are files and which are files that function as directories.

4.2.4 New features in the File System Editor

4.2.4.1 New items on File menu: Load, Edit

The File menu now has two new items, [Load] and [Edit].

4.2.4.2 Directory menu: wildcard deletion

The directory menu now has wildcard deletion. Use [Wildcard Delete], which prompts you with a default for deleting everything for the line that the menu applies to. It merges what you enter with the * defaults. It lists the files it intends to delete, asks for confirmation, deletes them reporting any errors, and updates the display.

4.2.4.3 Homedir on any host

Tree Edit Homedir has a new option on its mouse-R menu. It prompts for a host instead of using only the "logged-in" host (the one designated during login).

4.2.4.4 Free Records

[(mouse-M) Free Records] creates a directory-by-directory usage report of the file system. It now prompts you for a file spec for storing the usage report:

```
Filename for output listing:
```

4.2.4.5 Retrieving files from backup tapes

More actions are available from [Reload/Retrieve].

- Clicking left retrieves files from a backup tape.
- Clicking middle searches all the binary tape maps at the site for the tape locations of all backup copies of a file. It prompts for names, which it parses with respect to the local host unless you provide an explicit host name. All of the files requested must be from the same host. This applies only to Lisp Machine hosts. It looks at the binary tape maps on the specified host in the directory >dump-maps.

```
Enter file pathnames for which to search, separated by commas.
Wildcard names are allowed.  The default host is LOCAL-LISPM.
Paths: f:>sys>io>*.lisp, f:>bsg>*.init, f:>lisp>*.q*
Searching for:
  F:>sys>io>*.lisp.*
  F:>bsg>*.init.*
  F:>lisp*>*.q*.*
-----
F:>LISPM>COLDLD.QFASL.39, created 2/17/82 00:26:58, on tape fscns001
  (Backup dump of 2/24/82 13:05:44)
F:>LISPM>COLDUT.QFASL.57, created 2/18/82 19:06:36, on tape fscns001
  (Backup dump of 2/24/82 13:05:44)
F:>LISPM>EVAL.QFASL.13, created 2/19/82 04:45:17, on tape fscns001
  (Backup dump of 2/24/82 13:05:44)
....etc
----- Scan complete.
```

The default for all pathname components other than host is **:wild**, which matches anything; **:newest** is meaningless in this context.

4.2.5 Dumper takes multiple pathnames

In previous releases, the dumper prompted for a starting directory. Due to some special case checking, a file spec of ">foo" used to be interpreted as a directory, resulting in the files ">foo>*.*.*" being dumped. In other cases, the file spec would be interpreted as a (potentially wild) specification of what to dump. This inconsistency has been removed. The dumper's file specs are now always (potentially wild) specifications of what to dump.

The dumper now uses "Pathnames:" as the prompt instead of "Starting Directory:". It now accepts more than one file spec, with commas as separators:

```
Starting Directory: >sys>sys2>,>sys>backup>prev>*.lisp
```

The file specs are interpreted as they would be anywhere else in the system. All file specs are merged with local:**>*.*.* to form the pathname. From the example:

File spec	*Pathname*
>sys>sys2>	f:>sys>sys2>*.*.*
>sys>backup>prev>*.lisp	f:>sys>backup>prev>*.lisp.*

In particular, note that you need to supply the trailing directory separator ">" now, whereas you

did not need to do that previously. (Previously ">foo" would have been interpreted as a directory; this was in conflict with the syntax requirements of file specs elsewhere in the system.) Now,">foo>" is interpreted as a directory component; ">foo" would cause >foo.*.* to be dumped.

4.2.6 Dumper keyword arguments changed

The following changes were made to the keyword arguments in the dumper.

:pathnames is a new keyword. It accepts one or a list of pathnames or strings that are coercible to pathnames. It coerces these to pathnames, parses them against the local host as a default, and merges this with local:**>*.*.*. The result is then used as a list of starting pathnames to dump.

:start-path and **:start-node** have been changed. They are now equivalent to using **:pathnames** with the argument consisting of a list of the argument to the keyword.

:query is a new keyword. Its default is **nil** for standard operation. With the value **t**, it causes the dumper to offer its menu immediately instead of attempting to validate its arguments.

4.2.7 Backup dumper recovery

For an irrecoverable write error on a backup tape, the dumper tries to write an end-of-tape mark. It then asks for a new tape and redumps the files that are on the bad tape. It notes this in the dump map.

4.2.8 Backup dumper Interface change

At the end of its pass, the backup dumper used to leave the tape in the tape drive without rewinding it. Now the dumper's menu has a new item that lets you specify what to do with the tape at the end of the pass.

```
Tape when done: Offline Rewind Leave Query
```

Offline — Puts the tape offline and rewinds it. It declares the dump finished. This is the default.

Rewind — Rewinds the tape at the end of the pass without setting it offline. It declares the dump finished. This allows you to inspect the tape (using [(mouse-L) Reload/Retrieve / List Contents].

Leave — Leaves the tape positioned at the end without rewinding it or putting it offline. It declares the dump finished. It allows you to invoke the dumper again to append to the tape (using Append in that later dumper invocation).

Query — ~ Presents the following set of options.

```
Offline and Rewind, rewind, leave at end, or more files? (O, R, L, or M)
```

These are the same options and have the same meaning as if you had specified them originally. The meanings of the options are as follows:

- O Rewinds the tape and puts it offline.
- R Just rewinds the tape.
- L Leaves the tape at the end.
- M Lets you make another pass, asking you what files to dump in this pass.

You just specify one of these; no default is available here.

Using [Abort] terminates the dump normally, as if Leave had been specified, leaving the tape positioned for appending by a later dumper invocation.

4.2.9 Backup dumper map

When the dumper finishes a tape or finishes a pass, it leaves the dump map consistent and readable. Now, even if the machine subsequently crashes (before something that declares the dump finished), the dump map is available and can be read.

4.2.10 Salvager interface changes

The Salvager now offers a menu of options instead of having just a single prompt. The second option on the menu is a new feature, called Check Records, that verifies the records in each file and notifies you about any problems.

Some messages have been revised in anticipation of the multiple partition feature.

4.2.11 Maintenance menu changes

The maintenance menu has several new items.

Expunge All FS	Expunges each directory in the local file system. It displays a message about how much space was recovered.
Check Records	Verifies the records in each file and notifies you about any problems. (This is also available from the Salvager.)
Edit FSPT	Not to be used. Installed in advance of the multiple partition feature.

4.3 Improvements

4.3.1 Version management properties

The file system has two properties for managing versions of files: **:dont-reap** and **:dont-delete**.

Property	*Meaning*
:dont-delete	Means not to allow the file to be deleted. Any attempt, whether manual or automatic, to delete such a file causes an error. Files with this property are flagged with `@` in Dired and FSEdit.
:dont-reap	Prevents the automatic directory cleanup tools from deleting a file. That is, files with this property cannot be deleted by the generation retention count mechanism, by the directory cleanup tools in Dired, or by Reap File or Clean Directory in Zmacs. You can delete such files explicitly, manually. Files with this property are flagged with `$` in Dired and FSEdit.

4.3.2 Keyword change: :dont-reap

In previous systems, **:dont-reap** was a user-extension property. Now this property has been reimplemented as a standard property in the Lisp Machine file system.

4.3.3 Removing user properties in FSEdit

In some previous systems, FSEdit had problems with removing a user property from a file. You can now always remove a user property by setting its value to a null string.

5. Zmacs

5.1 Incompatible changes

5.1.1 Zmacs internal reorganization

The internal structure of Zmacs has been redesigned; some internal structures were reimplemented using flavors. Most changes are not visible to users, except those users who write editor extensions.

Functions that depend on an up-to-date parse of a Lisp buffer now work more reliably. In particular, Compile Changed Functions and Edit Changed Functions have been improved. Zmacs can now notice changes to definitions in the buffer and records when the definitions were last compiled or evaluated.

5.1.1.1 Incompatible changes

Zwei now implements intervals using flavors. Any editor extensions that used intervals might need to be converted to use flavor instances. They certainly need to be recompiled.

File buffers now have another tick so that the editor can distinguish the time when the file was last saved from the time when the file was first read in.

5.1.1.2 New features

Support buffers are a major new feature. These are buffers used by various support functions of the editor:

- Edit Buffers.
- View File.
- Lists for Edit Definitions, when more than one definition exists.
- Buffers for Dired and List Directory.
- Tags table commands.
- Everything that edits a sequence of definitions, as in List Callers or List Methods.

The lists created by these commands are now saved in support buffers. This means that you can examine the buffers containing the lists even after you have done some editing.

To avoid proliferation of editor buffers, Zmacs reuses support buffers in most cases.

5.1.1.3 Improvements

Each time you use a command that generates a set of possibilities (for example, Edit Callers and List Methods), it pushes the set of possibilities onto a stack. The set is popped from the stack when no more items remain in it. Several new informational messages are associated with this facility. When the whole possibilities stack is empty and you have nothing more pending it displays:

```
No more sets of possibilities
```

Suppose you had been using c-. to move through the set provided by Edit Callers and you then used Edit Methods to push a new set of possibilities onto the stack. When you finished the set provided by Edit Methods, you would see a message like the following to notify you that the empty set had been popped off the stack and the set of possibilities for Edit Callers had been reinstated.

```
c-. is now Edit Callers
```

The position of point in the support buffer indicates the next item for Next Possibility (c-.). You can select the support buffer and move point manually in order to skip or redo possibilities. The command now accepts a negative argument to pop a set of possibilities off the stack. An argument of zero selects the current support buffer of possibilities.

Compile Changed Definitions and related commands now work correctly. They no longer occasionally miss a definition that has changed or been added, or reprocess functions that have not changed.

Apropos is faster. It now reports key bindings for commands more quickly.

5.1.2 Commands containing "defun" and "function" renamed

The names of some commands and internal functions were changed to facilitate documentation and to more closely reflect their purpose. The command names that used to contain the word "defun" now contain "definition". For example, the former End Of Defun command is now called End of Definition. Some command names that used to contain the word "function" now contain the word "definition". The internal functions associated with these have also been renamed accordingly. See the table of changed commands on p. 83.

5.1.3 Change to buffer name completion

In Zmacs, buffer names are long strings that indicate the host and version as well as the file name. Hence completion is a necessary aid for entering buffer names. Previously, Select Buffer (c-X B) would take the string you entered and try to complete it to an existing buffer name. When completion was successful, it would select that buffer. Otherwise it would create a buffer by that name. This design was for compatibility with EMACS, which formed its buffer names differently.

Now it no longer creates a new buffer automatically. You have two ways to create a new buffer.

- You can supply a numeric argument to the c-X B command in order to have it create a new buffer.
- You can enter the buffer name with c-RETURN instead of with RETURN.

5.1.4 Change to command name completion

Command name completion in the case of a single trailing space has been changed. This applies only to the case where you use RETURN, not to the case where you use ALTMODE to see the completion followed by RETURN. It now trims the trailing space. This removes the confusion that occurred when one command name was the same as the initial substring of another, for example, Source Compare and Source Compare Merge. The following command now completes to Source Compare instead of to Source Compare Merge:

```
m-x so SPACE co SPACE RETURN
```

The following commands still complete to Source Compare Merge.

```
m-x so SPACE co SPACE ALTMODE
m-x so SPACE co SPACE END
```

5.1.5 Fundamental major mode is now default

The default value for major mode in the editor is now Fundamental Mode. The variable that controls this, for init file purposes, is **zwei:*default-major-mode***. The values that this variable takes have been changed. The new acceptable values are keyword symbols for the names of the modes. For example,

```
(login-forms
  (setq zwei:*default-major-mode* ':text)
  (setq zwei:*default-major-mode* ':lisp))
```

5.1.6 Default value for *default-package* is user

The default value for **zwei:*default-package*** is now the **user** package instead of the package from the previous buffer. The old behavior can be reinstated by setting this variable to **nil**. You can have any package as the default package, simply by specifying it as the value of the variable. For example, in your lispm init file:

```
(login-forms
  (setq zwei:*default-package* (pkg-find-package "tv")))
```

5.1.7 Zmacs command name and key changes

Some Zmacs commands have been renamed according to the following general rules:

- "Defun" became "Definition" in most cases and "Region" in a few cases.
- "Q Register" became "Register".
- "Function" became "Definition" where appropriate.

In cases where the command had a standard key binding, the key binding usually stayed the same; in these cases, you probably will not notice the name changes. A few key bindings were changed; see the list on page 84.

Previous name	*New name*
Add Patch Buffer Changed Functions	Add Patch Changed Definitions Of Buffer
Add Patch Changed Functions	Add Patch Changed Definitions
Beginning Of Defun	Beginning Of Definition
Compile Buffer Changed Functions	Compile Changed Definitions Of Buffer
Compile Changed Functions	Compile Changed Definitions
Compile Defun	Compile Region
Edit Buffer Changed Functions	Edit Changed Definitions Of Buffer
Edit Changed Functions	Edit Changed Definitions
End Of Defun	End Of Definition
Evaluate Buffer Changed Functions	Evaluate Changed Definitions Of Buffer
Evaluate Changed Functions	Evaluate Changed Definitions
Evaluate Defun Hack	Evaluate Region Hack

Evaluate Defun Verbose	Evaluate Region Verbose
Evaluate Defun	Evaluate Region
List Buffer Changed Functions	List Changed Definitions Of Buffer
List Changed Functions	List Changed Definitions
List Matching Functions	List Matching Symbols
List Q Registers	List Registers
Macro Expand Sexp	Macro Expand Expression
Mark Defun	Mark Definition
Point To Q Register	Save Position
Q Register To Point	Jump To Saved Position
Variable Document	Describe Variable

The following existing commands were assigned to keys or had their key bindings changed.

Key	*Command*
c-m-sh-E	Evaluate Region Verbose
c-sh-C	Compile Region
c-sh-E	Evaluate Region
c-sh-M	Macro Expand Expression
c-X J	Jump To Saved Position
c-X S	Save Position
m-sh-C	Compile Changed Definitions of Buffer
m-sh-E	Evaluate Changed Definitions of Buffer
m-X	Add Patch Changed Definitions
m-X	Add Patch Changed Definitions of Buffer
m-X	Evaluate And Replace Into Buffer
m-X	Evaluate Region Hack
m-X	List Matching Lines
m-X	List Matching Symbols

5.1.8 Dired subcommands changed

Several Dired subcommands have been reassigned to keys.

Key	*Meaning*
C	Copies the file on the current line.
R	Renames the file on the current line.
=	Compares the file on the current line with the newest version.

5.1.9 Word delimiter meaning changed for some characters

The following characters have been changed from delimiter syntax to alphabetic syntax in all modes except Midas Mode.

% $ period

The word motion commands no longer skip over these commands when they are embedded in words. This makes it easier to edit file names with the word commands.

5.1.10 Prefix character commands are not case-sensitive

A plain character following a prefix character command (like c-X) is now converted to upper case. This enforces the policy that, for example, c-X f is equivalent to c-X F.

5.2 New features

5.2.1 New Zmacs commands

5.2.1.1 Create Link

The Create Link command creates a link to a file. It prompts for the file name to link to in the minibuffer, using the standard conventions for defaults.

5.2.1.2 Create Directory

The Create Directory command creates a new directory. It prompts for a directory name, using the standard conventions for defaults. For consistency between hierarchical and nonhierarchical file systems, you specify the directory to be created as the directory component of a pathname. That is, you must end the directory name with whatever delimiter or separator is appropriate for the host. For example:

Host	*Directory string*	*Result*
TOPS-20	<A.B.C>	Creates directory C
Multics	>udd>Foo>Bar>z>	Creates directory z
Lisp Machine	>foo>bar>b>	Creates directory b
UNIX	/usr/jek/new/	Creates directory new

Currently, the file servers for VAX/VMS and TOPS-20 can fail to create directories, due to missing options.

5.2.1.3 Decrypt Buffer

Decrypt Buffer decrypts the contents of an encrypted buffer. It prompts for an encryption key, which does not echo as you type it. The encryption key given for decrypting must match the one used for encrypting. This command assumes the NBS encryption algorithm (also used by the Hermes mail system on various ARPANET hosts). See also section 5.2.1.6, p. 86.

5.2.1.4 Edit Buffers

Edit Buffers is similar to List Buffers (c-X c-B). The buffer listing that is produced by Edit Buffers is a buffer in its own right. It contains one line for each of the buffers in the editor. The lines are not currently mouse-sensitive. With the cursor on the line for a buffer, the following single character commands apply to that buffer.

RUBOUT	Undeletes buffer above the cursor.
SPACE	Selects that buffer immediately.
D	Marks the buffer for deletion (K, c-D, c-K are synonyms).
U	Undeletes either the buffer on the current line or the buffer on the line above.
S	Marks the buffer for saving.
~	Marks the buffer for setting not modified.
X	Executes an extended command (same as m-X).

You can install this command on a key using Set Key or by putting the following into your init file:

```
;;;add to Zmacs c-X comtab, NOT *standard-control-x-comtab*!!
(login-eval
  zwei:(set-comtab-return-undo
         *zmacs-control-x-comtab*
         '(#\c-B com-edit-buffers)
         ()))
```

5.2.1.5 Enable Host Capabilities, Disable Host Capabilities

The commands Enable Host Capabilities and Disable Host Capabilities provide a user interface from the editor to the new functions **fs:enable-capabilities** and **fs:disable-capabilities** (see section 2.2.20, p. 38).

These commands prompt for a host name. They choose the default host to offer according to the pathname for the current buffer. With a numeric argument, these commands also prompt for specific capabilities to enable or disable. The prompt lists any default capabilities for the host.

5.2.1.6 Encrypt Buffer

Encrypt Buffer encrypts the contents of the buffer. It prompts for an encryption key, which does not echo as you type it. It prompts twice for the key to ensure against typing errors. It uses the NBS encryption algorithm (also used by the Hermes mail system on various ARPANET hosts). See also section 5.2.1.3, p. 85.

5.2.1.7 Find File In Fundamental Mode

The command Find File In Fundamental Mode creates a fundamental mode buffer containing the file. This is useful because Zmacs does not parse the file while reading it in. Thus the names of the functions in the file do not conflict with those already known to completion in `m-.` and similar commands.

You can use Visit File (`c-X c-V`) in a buffer that is not associated with a file to get the same effect.

This replaces the previous pseudo-file-buffers facility.

5.2.1.8 Macro Expand Expression

Macro Expand Expression uses macroexpand on the current form. That is, it expands only the top level. This command is now available as `c-sh-M`.

5.2.1.9 Macro Expand Expression All

Macro Expand Expression All calls **macro-expand-all**. It fails in some cases because it cannot distinguish between macros and symbols in clauses in special forms (like **do**). This will be fixed in a future release.

5.2.1.10 Source Compare Merge subcommand

Source Compare Merge now has an "I" subcommand. It means to leave both alternatives in the text, along with the message lines from the source compare (`*** MERGE LOSSAGE ***`, and the like).

Source Compare Merge has a little-known mouse interface. You can answer the first question by

clicking left on the text you want to keep or on the dividing line between them to keep both. You can answer the second question by clicking Left for "yes" or Middle for "no".

5.2.1.11 View File Properties

View File Properties displays all the properties about a file that are maintained by the file system on which it resides. These are the properties like creation date and time, author, time of last access, and so on. For files on a Lisp Machine file system, it displays user-defined properties as well.

It prompts for a file spec, which it merges with the current default to form the pathname. Wildcards are not accepted; this must correspond to a unique file or directory name.

5.2.2 New or Improved Variables

The following variables can be set with Set Variable. In addition, they can be set in init files by using the internal form of their names. For example, Region Marking Mode is **zwei:*region-marking-mode*** internally.

Find File Not Found Is An Error

Value: **nil**. This variable controls whether trying to find a file that does not exist is an error. It applies to Find File and Find File In Fundamental Mode.

Value	*Meaning*
nil	Trying to find a file that does not exist creates a new file. Zmacs notifies you in the echo area with the message "(New file)". This is the default.
not **nil**	Trying to find a file that does not exist is an error.

Find File and Find File in Fundamental Mode now accept a numeric argument that means the same thing as having Find File Not Found Is An Error set to **nil**.

Region Marking Mode

Value: **:reverse-video** for setting the region to reverse video. The default is **:underline**. Reverse-video region display has been improved. It no longer has a band of white between the lines of the region.

Region Right Margin Mode

Value: **t**. Causes whatever marks the region (reverse video or underlining) to extend across unfilled space to the right margin. The default is **nil**.

One Window Default

zwei:*one-window-default* controls which window remains selected after a One Window (c-X 1) command when you were using more than one window. In prior systems, the default would have been **:top**. The default value in Release 4.0 is **:current**. Put the following into your init file to reinstate the old behavior.

```
(login-forms
  (setq zwei:*one-window-default* ':top))
```

Possible values:

:current
:other
:top
:bottom

This feature operates best when the current layout has no more than two windows. The value **:current** is the only one that is always well defined with more than two windows on the screen.

Check Unbalanced Parentheses When Saving
: This variable controls whether Zmacs checks a file for unbalanced parentheses when you are saving the file. The check is off (**nil**) by default because it slows saving down. When it does check a file that you are saving and finds unbalanced parentheses, it queries you about whether to go ahead and save anyway. This applies to all major modes based on Lisp; it is ignored for text modes.

zwei:*indent-new-line-new-line-function* *Variable*

This variable can contain a function for controlling what happens when you press LINE in an editor. (Normally it uses the functions bound to RETURN and TAB.) When this variable is not **nil**, it treats the value of the variable as a function to be executed in place of the function bound to RETURN.

zwei:*indent-new-line-indent-function* *Variable*

This variable can contain a function for controlling what happens when you press LINE in an editor. (Normally it uses the functions bound to RETURN and TAB.) When this variable is not **nil**, it treats the value of the variable as a function to be executed in place of the function bound to TAB.

5.3 Improvements

5.3.1 Improved Zmacs commands

Some Zmacs commands have been extended to take advantage of the wildcard facility. In particular, Delete File, Copy File, and Rename File now accept wildcard file specs.

Delete File
: Deletes one or more files. For a wildcard file spec, it lists the files that would be deleted and requires that you confirm the list. It deletes the files, showing any errors that occur but continuing rather than halting.

Copy File
: Copies one or more files to new files. If the source file spec is wild, the target file spec must also be wild. (See the sections on **copyf**, p. 66 and wildcard pathname mapping, p. 27.) For example:

```
M-X Copy File                       ;to Zmacs
  Copy File from:
     scrc:<lmfs>*.l*sp;0            ; All the newest .LISP and .LSP's
  to:
     ff:>sys-hold>scrc-sources>old-*.*.*

     SCRC:<LMFS>TEST.LSP.3 is copied into
     ff:>sys-hold>scrc-sources>old-test.lisp.3

     SCRC:<LMFS>FILES.LISP.147 is copied into
     ff:>sys-hold>scrc-sources>old-files.lisp.147
```

Note that .LSP gets mapped into .lisp because it uses canonical types when the type of the target pattern is **:wild**.

The canonical type mechanism determines the byte size and "type" of copy (Binary, Characters, and so on). For example, a file with a known canonical type of **:press** would be copied in 8-bit binary mode. For files that are not classified as binary files, the source file is opened in its default mode and the byte size is taken from the file system where it resides. The command uses **copyf** internally; see section 3.2.20, p. 66 for rules on determining byte size and transfer mode.

This command can now copy file authors and creation dates, when the target operating system supports setting these attributes. This action is not the default. The following numeric arguments control these aspects of copying. This command now accepts more numeric arguments for controlling characteristics of the copying. (Use HELP D for full details.)

Value	*Meaning*
7	Copy creation date and author.
8	Copy creation date and author, forcing binary copy.
9	Copy creation date and author, forcing character copy.

For hosts that do not support changing file creation dates, you must use this command with the argument 4, which inhibits copying the creation date.

Edit Changed Definitions

Determines which definitions in any Lisp mode buffer have changed and selects the first one. It now accepts a numeric argument to control the time point for determining what has changed:

Value	*Meaning*
1	For each buffer, since the file was last read (the default).
2	For each buffer, since the buffer was last saved.
3	For each definition in each buffer, since the definition was last compiled.

Several other commands now accept numeric arguments with similar meanings. See the on-line help for these commands.
Edit Changed Definitions of Buffer
List Changed Definitions
List Changed Definitions of Buffer

Rename File — Renames one or more files. If the source file spec is wild, the target file spec must also be wild. (See the section on wildcard pathname mapping, p. 27.)

5.3.2 Zmacs displays numeric arguments

Numeric arguments to commands appear in the echo area when a delay in typing the command occurs. With no delay, the argument does not appear.

5.3.3 Messages during compiling and evaluating

When an expression is being compiled or evaluated, the editor displays a message that classifies what is being compiled. For example,

```
Compiling Function FOO
Evaluating Variable BAR
Compiling Flavor MOO
```

It classifies macros and substs as functions (because all of these go in the function cell of a symbol).

5.3.4 + flag for buffers

Several commands that show buffers use the + character to mark new files that have not been saved. In addition, they now use + to mark new buffers, not associated with files, that have text in them. This is to support people who put text into a new buffer and later want to be reminded to write that buffer to a file.

5.3.5 Buffer and file attributes

The attributes for buffers are now stored separately from the attributes for the buffer's pathname. Previously, attributes were stored as a property of the generic pathname for the buffer so that the attributes would be independent of version number and as part of the buffer text. Now, attributes are stored as a property of the generic pathname, as part of the buffer data structure, and as part of the buffer text. As a result, it is now possible for the various copies to be different, if that is desirable.

The commands Reparse Attribute List and Update Attribute List help with keeping the information consistent. Reparse Attribute List copies the attributes from the buffer text into the generic pathname and the buffer data structure. Update Attribute List copies the attributes from the buffer data structure to the other locations.

One troublesome attribute is the package attribute. Information about it exists in four places. The Set Package command asks you which copies of the package information to update, using the following query:

```
Set it for the file and attribute list too?
```

Your answer affects the various versions of the package attribute as follows.

Location	*"Yes"*	*"No"*
Generic pathname	changes	same
Buffer property	changes	changes
Buffer text	changes	same
Current package	changes	changes

5.3.6 "Set" commands for file and buffer attributes

Each of the file attributes now has a set command associated with it. You have two choices when you want to change an attribute for a file:

- Edit the text of the buffer and then use Reparse Attribute List.
- Use the relevant set command and answer "yes" to its query. The meanings for "yes" and "no" are the same as for the Set Package command (except that only the Set Package command affects the current package).

The Set commands use the value of the variable **zwei:*set-attribute-updates-list*** to determine whether to query you about updating the file attribute list. The default value for the variable is **:ask**.

Value	*Meaning*
:ask	Always asks whether to update the attribute list.
nil	Never updates the attribute list.
t	Always updates the attribute list.

The following list shows the attributes for files, their associated set commands, and the default value for the attribute.

Attribute	*Set command*	*Default value*
Backspace	Set Backspace	**nil**
Base	Set Base	8
Fonts	Set Fonts	**nil**
Lowercase	Set Lowercase	**nil**
Nofill	Set Nofill	**nil**
Package	Set Package	**user**
Patch-File	Set Patch File	**nil**
Tab-Width	Set Tab Width	8 characters
Vsp	Set Vsp	2 pixels

The following table describes some of the attributes.

Backspace

Backspace controls whether a backspace character in a file displays as the word "overstrike" with a lozenge around it or performs the backspace. The default is the lozenge form.

Base

Base specifies the value of **ibase** that the Lisp reader is to use when reading forms from the file. Thus, Base controls the **ibase** used when you evaluate or compile parts of the buffer. This value does not affect the values of either **base** or **ibase** in the Lisp Listener you get by using BREAK.

Lowercase

Lowercase means that the file being edited is intended to contain lower-case code. When the Lowercase attribute is **nil**, whatever the user wants in the way of case handling prevails. People who want automatic upper-case code would use the following:

```
(login-forms
  (setq zwei:lisp-mode-hook 'zwei:electric-shift-lock-if-appropriate))
```

When the Lowercase attribute is anything but **nil**, the Electric Shift Lock Mode is never turned on automatically.

Nofill

When the Nofill attribute is **nil**, whatever you want in the way of autofilling behavior prevails. When Nofill is anything else, it means that autofilling is not appropriate for people who specify the mode of "autofilling if appropriate". The default is **nil**.

Use Nofill sparingly. Setting it means that everyone who edits the file has to be satisfied with Auto Fill Mode being off by default. In most cases, it is more reasonable to let an individual user's preferences prevail. It is useful for files that are not plain text, like mailing.lists, where you need to avoid spurious line breaks.

People who want to have auto filling available by default should use the following:

```
(login-forms
  (setq zwei:text-mode-hook 'zwei:auto-fill-if-appropriate))
```

People who don't want it never get it by default.

Patch-File	Patch-File means that the file contains patches. When a file is classified as containing patches, **fdefine** does not warn about functions being redefined during loading. Classifying something as a patch file also affects Edit Definition (which prefers files that are not patches) and **defvar** (which becomes **setq**). The default value is **nil**, meaning that the file does not contain patches.
Tab-Width	Tab-Width specifies how many spaces the editor uses between "tab stops". The default is 8 characters.
Vsp	Vsp is the vertical spacing (in pixels) between the text lines of an editor window. It specifies the distance between the descenders of one line and the ascenders of the next. The default value is 2.

5.3.7 Two window mode uses previous buffer

Using Two Windows used to create a new empty buffer (BUFFER-2) for the bottom window, unless you specified an argument. It now uses the previously selected buffer instead of creating many empty buffers.

5.3.8 Errors noted in file attribute lists

Zmacs warns you when it finds an unknown attribute in a file attribute list. It goes ahead and uses the unknown attribute in the list. The purpose of the warning is simply to help you detect misspellings.

You can define your own new file attributes, with the following form:

```
(defprop mode t fs:known-file-attribute)
```

See the file SYS: IO; OPEN LISP for models of defining file attributes.

5.3.9 Motion commands now use zwei:set-centering-fraction

More of the motion commands in the editor now use **zwei:set-centering-fraction** to determine how to reposition the window after cursor motion moves you past either the top or the bottom of the current screen.

5.3.10 User-defined major modes

In Zmacs, you can define your own major modes (see **zwei:defmajor** in the code). When you define the major mode, you can give it a property that indicates what type of major mode it is. This helps to set up a major mode that is a modification of one of the standard editing types. What you are telling Zmacs by doing this is how to parse the contents of a file in that major mode. The current editing types are **:lisp** and **:text**.

```
(defprop lisp-mode :lisp editing-type)
(defprop lil-mode :lisp editing-type)
(defprop text-mode :text editing-type)
(defprop bolio-mode :text editing-type)
(defprop fundamental-mode :text editing-type)
```

5.3.11 File types and major modes

You can control the default major mode associated with a particular file type. For example, Zmacs sets the major mode to Lisp for files with type "lisp". The repository for this information is a list called **fs:*file-type-mode-alist***. Suppose you wanted to associate the file type "tex" with text mode:

```
(push '("tex" . :text) fs:*file-type-mode-alist*)
```

The **car** of an element should be either a canonical type symbol or a string when the type is not one of the known canonical types.

In addition, suppose you have files that would require Scribe mode, if Zmacs had such a thing. You can define a correspondence between two major modes, using a new variable called **zwei:*major-mode-translations***. It is an alist of major mode names, expressed as keyword symbols. For example:

```
(push '(:scribe . :text) zwei:*major-mode-translations*)
```

6. Zmail

6.1 Incompatible changes

6.1.1 Recompile Zmail init files that use zwei:search-within-msg

Any compiled Zmail init files with filters that reference the macro **search-within-msg** (produced by the Search menu item in filter creation) must be recompiled to be used with Release 4.0. If such a filter appears in one of the alists associated with the [Move] or [Keywords] commands, you must recompile it before trying to use Zmail, since those filters get invoked to update the mouse documentation. Use the procedure in the following example:

```
(compiler:compile-file "oz:<mine>zmail.lisp" "oz:<mine>zmail.init")
```

6.1.2 Changes in getting new mail

The meanings of the mouse clicks to [Get Inbox] have been reassigned.

Command *Meaning*

[Get Inbox] or `G` from the keyboard
: Gets the new mail (inbox) for the current buffer. It has no effect when a collection is current.

[(mouse-M) Get Inbox]
: Prompts you for an inbox name for the current buffer.

[(mouse-R) Get Inbox]
: Calls up a menu of possible buffers to get the new mail for.

6.1.3 Changed profile options

Required Subjects
: Zmail now requires subjects on outgoing mail by default. You can also change this by setting the variable **zwei:*require-subjects***.

Value	*Meaning*
t	Requires a subject and prompts for one before sending if no subject has been supplied. (A null string satisfies the requirement.)
nil	Does not supply or require a subject.
:bug	Requires a subject for bug reports.
:init	Supplies an empty Subject: field but does not require that it be filled in.

6.1.4 Changed command names

Many command names and menu items in Zmail have changed. This was done as part of an effort to clarify the terminology and concepts in Zmail. The new terms are defined in a new document, *Zmail Concepts and Techniques.*

In summary, the essence of the change relates to the following terms:

inbox	The file where your mail is delivered, before being processed by Zmail.
buffer	Messages associated with a particular mail file.
collection	Messages collected from a mail buffer for some purpose (for example, as a result of filtering).
sequence	An umbrella term for referring to both buffers and collections.

All command names and on-line documentation have been revised to be consistent with the terminology as it is applied in the Zmail document. Please refer to it for details.

In addition to the changes referred to above, the following command names have changed:

Previous name	*Name in Release 4.0*
Delete Referenced Msg	Delete Referenced Msgs

6.2 New features

6.2.1 New editing commands

Zmail now defines the commands m-< and m-> (Goto Beginning and Goto End) for moving around in the message pane.

6.2.2 New conversation commands

A *conversation* is a collection containing all the messages that *refer to* a particular message and all the messages that refer to it. One message refers to another when it satisfies one of the following conditions:

- It has an In-Reply-To: field (except those generated by MM, a message system on TOPS-20 and TENEX hosts, which cannot be recognized).
- It has a References: or Supersedes: field.
- Its text contains at least the Date: and From: fields from the other message, indented as if it had been yanked in.

Commands that define conversations look in the current sequence (by default). For each message found, it collects all the messages that it refers to and all the ones that refer to it. When nothing new can be found, the conversation has been defined.

Due to the fact that it is defined by a chain of references, a conversation could be split if you deleted and expunged some key message in a chain.

The variable **zwei:*reference-default-universe*** controls where Zmail looks for the messages. When the variable is **nil**, Zmail looks in the current universe. You can set the variable to **zwei:*loaded-files-indirect-universe*** to specify searching all mail buffers.

Select Conversation by References
: Defines a conversation and selects it as a collection. This command is very similar to Select References.

Delete Conversation by References

Creates a conversation based on the current message and deletes all of those messages. It always deletes the current message.

6.2.3 New Zmail profile options

Inhibit Background Mail Checks
: Values: Yes or No. The default is No. Controls whether Zmail checks for new mail in the background when the Zmail frame is exposed. With the value No, it uses the Zmail background process to check for new mail in the default inbox. The result of this is to keep the file server active even when it is not really being used. With the value Yes, Zmail inhibits the background mail checking.

Prune Headers of Yanked Messages
: Values: Yes or No. The default is No. Controls whether Zmail removes most of the header fields from messages yanked into replies. With the value Yes, it results in a minimal header on the included message. With No, you can always use c-X Y to prune headers from an individual yanked message.

6.2.4 New message fields implemented: BCC, FCC, BFCC, Encrypted

Zmail now recognizes and generates several new message header fields.

BCC
: For "blind carbon copies". The field contains recipient names. The recipients in a BCC field do not appear in the copy of the message that is delivered to the ordinary recipients; they do appear in the copy that is delivered to BCC recipients. (All BCC recipients see the names of all BCC recipients.)

BFCC
: For filing a "blind" copy of a message that is being sent. The recipients of the message do not see the BFCC field. For example,

```
BFCC: F:>JHW>MAIL>OUTGOING.BABYL
```

The file has to exist already; BFCC cannot result in a file being created.

FCC
: For filing a copy of a message that is being sent. The recipients see the field in the message. For example,

```
FCC: F:>JHW>MAIL>OUTGOING.BABYL
```

The file has to exist already; FCC cannot result in a file being created.

Encrypted
: For flagging the message as containing encrypted text. Zmail generates this header field itself when it is sending a message. The value of the field is the name of the kind of encryption that was used.

Fonts
: For flagging the message as containing fonts. (This is like the Fonts attribute of a file attribute list.) Zmacs generates this header field itself when it is sending a message that contains fonts.

6.2.5 Encryption available

Zmail now supports encryption. Commands are available both when you are composing mail and when you are reading mail. Encrypted messages contain a new header field to indicate that they contain encrypted text.

The command to encrypt a message draft is Encrypt Text. Use it after you have completed the message draft but before you send it. Zmail prompts for an encryption key that the recipient will have to provide in order to decrypt the message. It converts the draft to a form that you cannot read. Decrypt Text is also available for message drafts. Both of these commands appear on the draft editor menu.

Decrypt Msg displays an encrypted message as plain text, prompting for the encryption key. You are only viewing the plain text form. Use a numeric argument to store the plain text version in the mail file.

Text yanked by Forward and Reply prompts for a decryption key rather than yanking unreadable text.

At this time, the only encryption algorithm supported is the NBS algorithm, used by Hermes. Further additions are planned.

6.2.6 Fonts in messages

Zmail can now interpret messages that contain fonts. These are the same fonts that the editor uses in Set Fonts and the editor font commands. The addition to Zmail to make it handle fonts consisted of adding a Fonts: header field. Reading a message that contains fonts on some different mail system is somewhat tedious, due to the presence of the ^F font change indicators.

6.2.7 Internet domain addressing supported

Zmail understands the Internet RFC822 domain-addressing formats, in case you receive any mail in the Internet format.

6.2.8 New Zmail facility: zwei:preload-zmail

You can now specify in your init file a series of actions, related to starting up Zmail, that can run in the background. In order to use this feature from your lispm init file, you must first enable Zmail to run in the background. The flag that controls this is **zwei:*hang-background-process-when-deexposed***. This flag is normally **t**. You must set it to **nil**.

An example:

```
(login-forms
  (setq zwei:*hang-background-process-when-deexposed* nil))
(zwei:preload-zmail '(:load-file "f:>rwk>mail>garbage.babyl")
                    "f:>rwk>mail>gubbage.babyl")
```

This first causes the Zmail init file to be loaded and then loads the two files garbage.babyl and gubbage.babyl, in turn.

Action specs can take one of two forms:

Argument	
string or *pathname*	(Loads the file with that name)
:load-file *string-or-pathname*	
:hang-when-deexposed *t-or-nil*	

The string form is simply an abbreviation for **:load-file** followed by a string.

When the function is in your LISPM init file, it first enqueues a command to load your Zmail init file and then queues any action specification arguments. You can preload your default mail file; it does not become selected until you use [Get Inbox] or [Select].

The function can appear instead in your Zmail init file. In that case, nothing can happen in the way of preloading until you first select Zmail and use a command that initiates reading that init file.

Regardless of where the preload function occurs, it can include a keyword that reinstates the flag that shuts down the Zmail background when Zmail is not selected. This is the form **'(:hang-when-deexposed)**. Using this is equivalent to resetting the deexposed action flag to t.

Actions requested in the foreground (such as reading your default mail file) have priority over preload actions. Preloading does not attempt to reload a file that has already been loaded.

6.2.9 Adding bug lists to Zmail

You can now add a new bug-mail recipient to the list of bug recipients. Two mechanisms are available.

- Use the new **:bug-reports** option to **defsystem**.

  `(:bug-reports` *system-name documentation-string*`)`

  The documentation-string is the mouse-line documentation for the menu item. *system-name* appears on the Zmail menu. For example,

```
(defsystem print
  (:name "print")
  (:pathname-default "sys: print;")
  (:package print)
  (:patchable)
  (:not-in-disk-label)
  (:bug-reports "Print" "Report a bug in the hardcopy facility.")
  ...)
```

- Use this function:

  zwei:add-bug-recipient *name* &optional *documentation*
  zwei:add-bug-recipient adds a new recipient to the menu available from [(mouse-M) Mail]. Both arguments are strings. *name* appears in the menu; *documentation* appears in the mouse line documentation. This uses the site option **host-for-bug-reports** to determine the rest of the address.

6.3 Improvements

6.3.1 Zmail works with UNIX hosts

Zmail is now fully functional with UNIX. Zmail can both read new mail and add the new mail to the standard UNIX mail file, `mbox`.

Zmail can also handle BABYL format mail files on UNIX, using the following conventions for finding files:

inbox file	`/usr/`*user-id*`/mbox` and `/usr/spool/mail/`*user-id*
default mail file	*user-id*`.bb`

The BABYL mail file format allows status and keyword information to be stored with messages; the UNIX mail reading programs cannot read BABYL files.

To use *user-id*`.bb` as your default mail file, edit the options for the file to specify two files as inboxes for the BABYL file. For example:

```
Mail:/usr/jek/mbox,/usr/spool/mail/jek
```

6.3.2 Reference commands changed

The commands with the word "reference" in their names are now much faster. They were reimplemented to use hash tables rather than searching. With a numeric argument, the Reference commands now offer a menu of universes for searching.

6.3.3 Background process changed

The background process now sleeps when the foreground process is running a command or whenever you are typing in any window. Thus the background process never slows down the operation of the foreground.

6.3.4 In-Reply-To fields included

The defaults for creating reply messages have been changed so the In-Reply-To fields are now automatically included. The default can be changed by changing the option Generate In Reply To Field in the Zmail profile.

6.3.5 Changes to keyboard interface

c-F	Moves to the next message containing a specified string (using **zwei:com-zmail-find-string**).
F	Forwards the current message (using **zwei:com-zmail-forward**).
L	Labels the current message (using **zwei:com-zmail-keywords**).
O	Moves the current message to a file (using **zwei:com-zmail-move**).

6.3.6 Change to Local mail

Local mail is a facility for putting a message into a mail file without having to give it any recipients. (Otherwise, Zmail forces you to provide at least one To: recipient).

[Mail / Local] now uses **zwei:fcc-local-mail-template**. It initializes the message being composed with an fcc header to the file for the current mail buffer.

The variables ***local-mail-header-force*** and ***local-mail-include-subject*** are now obsolete. The template **compatible-local-mail-template** is also obsolete. Its function is still available, however, from [Mail / Local Mail / Compatible Local Mail].

6.3.7 Reply shows all header lines

The header window in a reply now shows all of the automatically generated header lines, making the header window larger if necessary.

6.3.8 Universes reimplemented

Universes were generalized and reimplemented using flavors. People with advanced Zmail extensions might find it necessary to rewrite some of these using the message **zwei:map-over-universe**. You can now define arbitrary universes; see `SYS: ZMAIL; UNIVERSE LISP` for information.

6.3.9 Change for ITS users

Zmail now allows a BABYL file to specify which host to use for **gmsgs**. The option is called **:gmsgs-host**. Thus if you keep your BABYL file on a host that does not have a **gmsgs** server, you can still have **gmsgs** service by specifying an ITS host.

6.3.10 Case is preserved in filter and universe names

Zmail now preserves the case that you type for filter and universe names, instead of converting them to upper case. Names that have already been defined are not affected.

This affects only people who are typing lower-case names as symbols. Remember to use the vertical bars around the name.

7. Notes and Clarifications

7.1 Clarifications and corrections

7.1.1 Loading files in the background

Loading files asynchronously is not guaranteed to work. You cannot load files, particularly patch files, in a background process and expect the correct results. Some reasons:

- The file could be doing something that maps over all pathnames, expecting that pathnames would not change while it was running.
- **defflavor** has no locking at load time. Thus, the flavor data structures can be damaged if two processes evaluate **defflavor** simultaneously.
- **load-patches** can reset and rebuild the site information.
- When a foreground bug occurs while patches are loading, you cannot determine what system the bug occurred in.
- When you are using a subsystem in the foreground while it is being patched in the background, unexpected problems could arise.

Making patch files is a difficult process; requiring patches to load asynchronously would make it significantly harder to create patches.

Besides the fact that it is not guaranteed, asynchronous loading would not be efficient. The main process and the background process would be competing for resources and you would lose a lot of time to paging and the scheduler.

7.1.2 Loading into packages

You cannot load a program into a package that has subpackages. This is why you cannot load a program into **global** or **system**; these packages were locked against interning new symbols in order to keep you from mistakenly trying to load a program into them.

Packages with subpackages should have symbols interned in them intentionally only. This is because the names are shared among the subpackages. Having symbols interned automatically by **read** and **load** makes things unpredictable. For example, you could never tell whether trying to intern the same name in two subpackages would produce two different symbols in two different subpackages or one symbol in the parent package. A program loaded into the parent package might use the name for a local variable, thereby causing the name to be interned and resulting in the two subpackages sharing the name.

7.1.3 Restriction enforced on defun-method

In *System 210 Release Notes*, p. 4, it states that **defun-method** requires a symbolic function spec as its argument. This restriction is now true; anyone who was depending on the fact that this restriction was not enforced in System 210 will have to change their code.

7.1.4 #X reader macro

#X followed by a number reads the number in radix 16 (hexadecimal). For numbers that contain A through F as "digits", you must also supply a plus sign or a minus sign. Otherwise, the number would be interpreted as a symbol.

```
#x+F  =>  15.
#x10  =>  16.
#x+10  =>  16.
(setq f "foo")
#xF  =>  "foo"                  ;F is a symbol
```

7.1.5 format directives for date and time

format accepts some directives for printing times and dates.

~\date Prints its argument as a date and time, assuming the argument is a universal time. It uses the function **print-universal-date.**

```
(format nil "Today is ~\date\" (time:get-universal-time))
=> "Today is Wednesday the twenty-second of September, 1982; 3:07:05 pm"
```

~\time Prints its argument as a time, assuming the argument is a universal time. It uses the function **print-universal-time.**

```
(format nil "Today is ~\time\" (time:get-universal-time))
"Today is 9//22//82 15:08:41"
```

~\datime Prints the current time of day. It does not take an argument. It uses the function **print-current-time.**

```
(format nil "Today is ~\datime\")
"Today is 9//22//82 15:19:06"
```

~\time-interval Prints the length of a time interval. It uses the function **time:print-interval-or-never.**

```
(setq a (time:get-universal-time))
...
(format nil "It is ~\time-interval\ since I set this variable"
        (- (time:get-universal-time) a))
"It is 1 hour 5 minutes 9 seconds since I set this variable"
```

7.1.6 Little-known command: View Mail

View Mail (m-X) is a command for viewing your inbox file. It uses the standard mail pathname for your home directory. When no new mail has been delivered recently, it reports "No new mail". This command uses View File.

7.1.7 Names for definitions: sys:function-parent

When a symbol's definition is produced as the result of macro expansion of a source definition, so that the symbol's definition does not appear textually in the source, the editor cannot find it. The accessor, constructor, and alterant macros produced by a **defstruct** are an example of this. The **sys:function-parent** declaration can be inserted in the source definition to record the name of the outer definition of which it is a part.

The declaration consists of the following:

(sys:function-parent *name type***)**

name is the name of the outer definition. *type* is its type, which defaults to **defun.** (This is the same type as in **record-source-file-name**; it is usually the name of the defining special form.)

You can define the type of an entity being defined:

```
(defprop feature "Feature" si:definition-type-name)
(defprop defun "Function" si:definition-type-name)
```

sys:function-parent is a function related to the declaration. It takes a function spec and returns **nil** or another function spec. The first function spec's definition is contained inside the second function spec's definition. The second value is the type of definition.

Two examples:

```
(defsubst foo (x y)
  (declare (sys:function-parent bar))
  ...)

(defmacro defxxx (name ...)
  `(local-declare ((sys:function-parent ,name defxxx))
     (defmacro ...)
     (defmacro ...)
  ))
```

7.1.8 Patches to "System"

Do not attempt to make patches to the system named "System" or to any of the other distributed systems. Local patches to the system are likely to be lost during the next update distribution. If you try to patch a distributed system, you receive a warning. See section 3.2.15, page 64 on making field patches to distributed systems.

7.1.9 Removing a defun-method

When you redefine a **defun-method** to no longer be a **defun-method,** you must use **undefun-method** for the **:defun-method** function generated internally by it. Otherwise the compiler will think that the function is still a **defun-method** and hence will generate the wrong code.

7.1.10 union and intersection use eq

The *System 210 Release Notes* define the functions **union** and **intersection.** It did not mention the fact that these functions use **eq** for their comparisons. You cannot change the function used for the comparison.

7.1.11 Package nil in make-system

In **make-system,** you can declare a package for the system with a **:package** declaration. Sometimes you have a module that needs to use the packages specified by the files' attribute lists rather than the package declared for the system. You make the files' package specs override the general one by putting **:package nil** in the module's plist (at the end of the **:module** declaration).

7.1.12 Clarification for process-wait-with-timeout

The description of **process-wait-with-timeout** in the *Lisp Machine Manual*, p. 431, reverses the sense of the returned value. The following description is correct.

process-wait-with-timeout *whostate interval function* &rest *arguments*

This is a primitive for waiting. It applies *function* to *arguments* until the function returns something other than **nil** or until the interval times out. *interval* is time in 60ths of a second. When the process times out, **process-wait-with-timeout** returns **nil**. When the function returns something other than **nil** within the interval, **process-wait-with-timeout** returns **t**.

7.1.13 Clarification of read errors and the rubout handler

Any error which occurs at *read time*, as opposed to eval time or compile time, does not take you into the Debugger. For instance, type "(a b ,c" to a Lisp listener, or "#$", or a floating point number with too many digits to the right of the exponent. Any error that occurs in the reader is trapped by the rubout handler so that you get a chance to edit your input and continue. The philosophy behind this is that your error is one of syntax and therefore should be obvious from what is on the screen, so there is no need to go into the Debugger. This explains why you do not see any restart handlers listed after this kind of error.

7.1.14 Clarifications for TOPS-20 users

The Zmacs command List All Directory Names lists subdirectories as well as top-level directories for TOPS-20 hosts. This is a result of the TOPS-20 command set. On a Lisp Machine file system, it lists only top-level directories.

The TOPS-20 file server contains a known bug. Very occasionally when you are writing out a file, you receive an error message including the phrase "data packet being discarded". Abort the write operation and start over. In our experience, the bug is not reproducible, which is why it is "known" and not fixed. Please save any reproducible case.

7.1.15 Clarifications for VMS users

The VMS file server does not support file name completion.

7.2 Practical advice

7.2.1 Garbage collector

The LM-2 garbage collector is a copying incremental garbage collector. It requires space in which to copy the structures, in addition to any space already in use. A background process called the Scavenger goes through all the objects in *old* space that are still in use and copies them into *copy* space. In the meantime, any new consing is done in *new* space. When all the structure which is reachable in old space has been moved into copy space, old space is reclaimed. Later on, when only enough room remains to copy the maximum amount of data, the *flip* occurs; New space and copy space are changed to old space and new new and copy spaces are started.

Until the scavenging process is complete, running with the garbage collector can require up to twice as much space as running without the garbage collector (depending on how much of old space was garbage, compared to how much had to be copied). If you have been running without the garbage collector for a long time, you might not have enough room to successfully run the garbage collector and collect all the garbage. The garbage collector sends notifications now as you approach a certain percentage full.

One solution is to turn on the garbage collector sooner, so it is left with enough space to operate. Another is to use **gc-immediately**.

Before the first flip, all allocated memory is new space; the Scavenger does not run until after the first flip.

Further information about improvements to the garbage collector utility appears in section 3.2.1, p. 56.

7.2.1.1 Strategy for unattended operation with the GC

It is chancy to leave very large compilations that do a lot of consing running unattended. In previous releases, the GC would sometimes simply halt the machine. Now you can set the following variables in order to control the assumptions that it makes about the amount of space needed or available (see section 3.2.1, 56).

si:gc-flip-minimum-ratio
si:gc-flip-ratio
si:gc-reclaim-immediately-if-necessary

Some people find it necessary to have garbage collection working in order to load large systems. People in these situations who do not want to use incremental garbage collection should be using **si:gc-immediately** rather than **si:full-gc**. **si:full-gc** does a lot of unnecessary work and disables multiprocessing, thus causing network connections to be lost. **si:full-gc** should be used only in conjunction with the band compressor.

7.2.2 Supplying a personal name for Finger

The TERMINAL F commands show the users logged in on various machines. For time-sharing machines, the protocol uses a user database to determine the personal names that match the user names. Currently for hosts that do not run this protocol or for sites that do not have any time-sharing hosts, users do not have any personal name information available. You can supply a personal name in your init file in these cases:

```
(login-forms
  (setq fs:user-personal-name-first-name-first "Billy T. Kid"))
```

7.2.3 One mouse click or two

Some applications programs use shifted mouse buttons for encoding special commands. These programs can be adversely affected by the change announced in *System 210 Release Notes, section 4.6.* That change made SHIFT, CTRL, and HYPER into shift keys for representing mouse double clicks.

Applications that assign their own meaning to modified mouse clicks should protect themselves by turning off the double-click meanings for these keys. Bind the value of

tv:*mouse-incrementing-keystates* to **nil** around any calls to **tv:mouse-button-encode**. The system calls **tv:mouse-button-encode** implicitly in the mouser process. You need to add a wrapper to **:mouse-buttons** if you inherit the standard default method.

7.2.4 Metering large computations

Q: I am using METER:TEST and METER:ANALYZE on a computation that runs for about half an hour and it doesn't seem to work.

A: The data base for **meter:analyze** very large and slow; it is impractical to meter anything that runs for more than a minute or so. Also unless you allocated a gigantic METR partition, **meter:test** probably filled up the METR partition in the first few seconds and discarded the rest of the data.

7.2.5 Only one compiler at a time

As previously announced (*System 78 Release Notes*, *System 210 Release Notes*), only one process at a time can use the compiler. Attempts to invoke the compiler while it is running produce a message like the following:

```
[10:29 Compiler in process ZMACS-WINDOWS:  waiting for resources.]
```

This means that you tried to run the compiler in the zmacs-windows process, but some other process is running in the compiler and is holding the global compiler lock. If you want to do your compilation, select the process that is using the compiler and either abort it or wait for it to finish. Your process that produced the error then wakes up and proceeds. Otherwise, you can give up on the attempt that produced the error by using c-ABORT on that process.

7.2.6 Adding mouse documentation in Choose Variable Values menus

Q: When you use Choose Variable Values (documented in *Lisp Machine Choice Facilities*) and you have a variable that can take on any of a small fixed set of values, naturally you use the **:choose** keyword as described on p. 15. However, that doesn't let you add mouse-documentation to the individual choices. How do you do that?

A: By using **:menu-alist** instead of **:choose**. **:menu-alist** is documented just two items down from **:choose**, but you could easily not notice it, which is why this question is asked frequently.

See the function **lmfs:fsmaint-fspart-init-menu** as a good example of how to use this kind of menu.

7.2.7 zwei:edit-functions helps with editing jobs

zwei:edit-functions *spec-list*

> **zwei:edit-functions** gives *spec-list* to the editor in the same way that Edit Callers and similar editor commands would. It is like **ed** in that inside the editor process it throws you back into the editor, whereas from another process it just sends a message to the editor and selects the editor's window.
>
> This command is useful when you have collected the names of things that you need to change, for example, using some program to generate the list. *spec-list* is a list of definitions; these are either function specs (if the definitions are functions) or symbols.

Zmacs sorts the list into an appropriate order, putting definitions from the same file together, and creates a support buffer called ***Function-Specs-to-Edit-n***. It selects the editor buffer containing the first definition in the list.

7.2.8 Making standalone editor windows

Q: I want to make an editor window with the following properties:

- Should be standalone (have its own process).
- Need not have the buffer structure of Zmacs.
- Need not even have minibuffers. If I must have one, I want the pop-up style.
- Needs a special comtab. That comtab will have commands that make the window do something worthwhile.

How do I do this?

A: Start with **zwei:standalone-editor-frame**. Send it an **:edit** message to make it edit. It does not have its own process by default; you can mix **process-mixin** with it and make that process send the **:edit** message if you want it to have its own process.

Two other useful messages:

:set-interval-string
Inserts a string in the editor.

:interval-string Returns a string to the caller when **:edit** returns.

For providing a special comtab, you can initialize the instance variable **zwei:*comtab*** by using the **:*comtab*** keyword in the init plist.

The user exits from this kind of editor by using END.

7.2.9 Layouts menu item

The Layouts item in the system menu does not work as you might hope or expect. Pending redesign, you cannot save any layouts or specifications for layouts in your init file.

7.2.10 Using char-upcase function

The **char-upcase** and **char-downcase** functions interpret bits in the **%%ch-char** and **%%ch-font** byte fields as font information rather than as modifier bits. (See *Lisp Machine Manual*, p. 315.) This could confuse you if you try to call these functions on fixnums that you yourself interpret as characters with modifier bits, particularly if you take keyboard input and call these functions. You should check characters received from **tyi** to determine whether modifier bits are present; use **char-upcase** only when no modifier bits are present.

The current Common Lisp design has character objects, with separate modifier and font information. The problem cannot be fixed before we adopt Common Lisp because it would cause compatibility problems.

7.2.11 Information about temporary consing areas

extra-pdl-area contains objects that are pointed to only by the local state of the machine (mainly the stack buffer). When this area becomes full, it is completely reclaimed and objects in it that are pointed to by the local state of the machine are copied out, and the pointers are relocated. The restriction that only the local state of the machine is permitted to point to the extra-pdl area is enforced in the following way. Any time a pointer that points into the extra-pdl area is stored into memory, the object it points to is copied out of that area and the pointer is changed to point to the copy.

The extra-pdl area works only for numbers, because when it copies out objects it cannot guarantee that they "remain eq to themselves". That is, it cannot guarantee that all pointers to the object are changed to point to the copy. This could be changed only at an efficiency cost.

The extra-pdl area is used to store those numbers that are not immediate: full-size flonums, bignums, and rationals.

The lifetime of objects in the extra-pdl area is very short. A stack-group switch necessarily flushes this area since it stores the entire local state of the machine into memory. Stack group switches happen at least once a second.

8. Operations and Site Management

8.1 Incompatible changes

8.1.1 Sys host logical directory changes

Due to internal reorganization, some logical directories have been renamed, some have been removed, and new ones have appeared. The following table explains the details. Some of these directories contain system sources and are required. Others are optional and need to be created only if you plan to load the corresponding software package.

System 210	Release 4.0
canon	lgp (required)
lbp	*none*
none	doc (required)
none	hardcopy (required)
none	lmfs-patch (required)
none	print (optional, for LGP only)
none	interlisp (optional, to be released soon)
none	mailer (optional, to be released soon)

8.2 New features

8.2.1 Installing the print spooler

Release 4.0 includes a print spooler for a Symbolics LGP-1 laser printer. The section describes how to install the spooler software.

This applies only if your LGP-1 is directly connected to an LM-2 at your site. Follow this procedure only after you have completed the installation of Release 4.0. Add the **:hardcopy-devices** site option, and the related site options (see section 8.2.2, p. 112) and update the site parameters at your site (see the Software Installation Guide) so that all of your LM-2's can transmit to the spooler.

The LM-2 that connects to the LGP-1 must include the print spooler in its Lisp environment. Note that the source files, qbin files, and patch files for the PRINT system are all provided on the standard Release 4.0 distribution tape. Follow these steps to create the necessary Lisp environment from a site-configured Release 4.0 band:

1. Cold boot and log in without running your init file.
2. `(make-system 'print)`
3. `(load-patches 'print ':noselective)`
4. `(disk-save` *n*`)`
5. `(set-current-band` *n*`)`

(See *Software Installation Guide* for an explanation of this procedure.)

You can install this band on another machine, even if that machine does not have an LGP-1 connected to it. The print spooler operates only when the LGP-1 is present.

The print spooler starts up automatically when services are enabled (see section 8.2.7, p. 116).

The print spooler contains the following functions:

print:shut-down-print-spooler
Turns off the spooler if it is running.

print:reset-print-spooler
Turns off the spooler if it is running, and starts up a new one.

print:print-queue
Displays the LGP queue just like **chaos:print-lgp-queue** but doesn't use the Chaosnet to do it.

print:print-log
Prints out a log of interesting server events, most recent first, to aid in debugging.

8.2.2 New site options for hardcopy support

Release 4.0 provides three new site options.

:hardcopy-devices
:default-hardcopy-device
:default-screen-hardcopy-device

:hardcopy-devices
Specifies the devices at the site. It is a list of device lists. Each device list contains the name of the printer as a string and a plist of options. The options specify what the hardware interface is and how spooling is done. Detail on the options appears on p. 113.

:default-hardcopy-device
Specifies by name the default output device for hardcopy commands. The name of the default device must appear as one of the entries in the **:hardcopy-devices** list.

:default-screen-hardcopy-device
Specifies by name the output device for screen copy commands. The name of this device must appear as one of the entries in the **:hardcopy-devices** list.

Using these site options requires a properly connected hardcopy device. In Release 4.0, this means a Symbolics LGP-1 printer, a Versatec, a Xerox Dover printer, or a locally connected serial-port printer.

Site options for two LGP-1 printers, each connected to a Lisp Machine, might look like this:

```
(:hardcopy-devices
  '(("Winnipesaukee" :format :lgp
                     :spooler :chaos :host "Pointer"
                     :contact-name "LGP" :interface :lgp
                     :spooler-home-directory "pointer:>print-spooler>")
    ("Quinsigamond" :format :lgp
                    :spooler :chaos :host "Afghan"
                    :contact-name "LGP" :interface :lgp
                    :spooler-home-directory "afghan:>print-spooler>")))
(:default-hardcopy-device "winnipesaukee")
(:default-screen-hardcopy-device "winnipesaukee")
```

The value of the **:hardcopy-devices** option is a list, where each element of the list describes one hardcopy device. The element starts with the name of the device, followed by keyword/value pairs that give interface information about the device:

(name keyword1 value1 keyword2 value2 ...)

The keywords specify what format the device accepts (**:format**) and how the device is accessed at the site (**:spooler** and **:interface**). **:format** is required for each device; at least one of **:spooler** and **:interface** must be provided for each device.

Keyword	*Values*
:format	Describes the format accepted by the device. The value is a keyword, one of the following:

Keyword	*Meaning*
:lgp	The device accepts Symbolics LGP-1 codes.
:press	The device accepts Press file format.
:xgp	The device accepts Xerographic Printer (XGP) codes.
:simple	The device is a simple printer, such as a hardcopy computer terminal or daisy-wheel printer and accepts ASCII characters.

:spooler Describes how to access the spooler that controls the device. This should be used if some spooler program accepts requests for the device. Do not use this keyword if the device is accessed directly; for that use **:interface** instead. The value is a keyword, one of the following:

Keyword	*Meaning*
:chaos	The spooler is connected over the Chaosnet. The **:host** and **:contact-name** options must be provided as well.
:eftp	The spooler is connected over the Experimental 3-MB Ethernet. The **:host-address** option must be provided.
:its-dover	The spooler is specifically the ITS Dover spooler at MIT. The **:file-name** option must be provided.
:gould	The spooler is for a Gould printer at MIT. The **:file-name** option must be provided.
:file	This means that the way to spool the request is to write a file with the filename specified by the **:file-name** option. The file is opened in binary mode with byte-size 8.
:ascii-file	This means that the way to spool the request is to write a file

with the filename specified by the **:file-name** option. The file is opened in character mode.

:interface Describes which host the printer is connected to and how it is connected by hardware to the LM-2. This keyword applies only when the printer is connected to the LM-2 by hardware and is accessed directly, not by a spooler. The value is a keyword, one of the following.

:serial The printer is connected over the serial line. In the case of **:interface :serial**, all the keyword options to **make-serial-stream** are available. They default the same as for that function, with the following exceptions:

Keyword	*Default*
:baud	**1200.** (rather than 300.)
:ascii-characters	**t**
:force-output	**nil**

Another useful keyword is **:xon-xoff-protocol** (C-S/C-Q). Most printers are Data terminals so your Lisp Machine needs to be a Data set. See *LM-2 Serial I/O* for more details.

:lgp The printer is a Symbolics LGP-1 connected via a DR-11 interface.

:host Specifies the name of the chaos host to connect to as a string. This option applies only under one of the following conditions:

The **:spooler** keyword with value **:chaos**
The **:interface** keyword without the **:spooler** keyword.

:contact-name Specifies the contact name as a string to use in opening the Chaos connection. This option applies only if you used the **:spooler** keyword with value **:chaos**.

:host-address Specifies the name of the Experimental Ethernet host to connect to as a fixnum. This option applies only if you used the **:spooler** keyword with value **:ether**.

:file-name Specifies the name of the file to write as input to the spooler. The value is a file spec string. This option applies only for the **:spooler** keyword used with one of the following values: **:its-dover, :gould, :file,** or **:ascii-file**. Sometimes only part of the file spec string is actually used:

Value	*Result*
:its-dover	The name, type, and version in this pathname are ignored; only the host, device, and directory are used.
:gould	The type and version are ignored; only the host, device, directory, and name do. The name is used to form the name of the file; see the documentation of the Gould spooler.
:file, :ascii-file	The entire file name is used. Using a type of **:newest** here is likely to be the right thing; otherwise, successive spool requests would overwrite one another.

:default-font Specifies the default font for the device for printing text files. The font is specified as a list of three elements naming the font, the face, and the size. The default font can be can be omitted although it is a good idea to specify it in case the defaults provided are not be available at your site.

:header-font Specifies the default font for the page headings when the device is printing text

files. The font is specified as a list of three elements naming the font, the face, and the size. The header font can be omitted although it is a good idea to specify it in case the defaults provided are not be available at your site.

:spooler-home-directory
Specifies the directory for temporary files needed by the print spooler (section 8.2.1, p. 111). Supply a string containing the host name and the directory name.

Example:

```
(defsite :my-site
  ...
  (:hardcopy-devices
    '(("Fred" :format :lgp :spooler :chaos :host "LM-1"
              :contact-name "LGP")
      ("George" :format :simple :interface :serial
                :host "LM-3" :xon-xoff-protocol t)))
      ("Itasca" :format :lgp :interface :lgp
                :spooler :chaos :host "cadr9"
                :contact-name "LGP"
                :default-font ("Fix" "B" 9.)
                :header-font ("lpt" "" 10.))
  ...
)
```

Without **:spooler**, the device can be accessed only from the LM-2 to which it is directly attached. In Release 4.0, the print spooler provided handles only the Symbolics LGP-1; it cannot spool files to serial-line printers. This restriction will be removed in a future release. Another restriction is that only screen images, not text, can be sent to a Gould spooler.

8.2.3 New site option: :fonts-widths

This option specifies to load the file containing the font width information at this site. Value t causes the site initialization list to try to load the following fonts file:

```
sys:press-fonts;fonts widths
```

Only sites with their own additional LGP fonts would set the value to t. The site font width information replaces the distributed font width information.

8.2.4 New site option: :supdup-default-path

:supdup-default-path is a new site option for specifying the default gateway host for **supdup** and **telnet**. Its value is the name of the host.

```
(:supdup-default-path "MIT-MC")
```

The gateway host is used only if you try to connect to a site on the Arpanet. the host must support the Chaosnet's Arpa Protocol.

8.2.5 New site option: :chaos-tape-server-hosts

All tape-using programs now call **tape:default-host** to supply a default tape host for **choose-variable-values** and other queries. (If you call **tape:make-stream** without specifying any host, it does this querying itself.) The default gets set by actually trying to open a tape stream.

The default is initialized at cold or warm boot to "Local" if you have a local tape, or the first item in the list if not.

```
;; Hosts with tape drives running servers for the Remote Tape Protocol (preference order)
(:chaos-tape-server-hosts '("scrc-pointer" "scrc-tenex"))
```

8.2.6 New Initialization lists: enable-services, disable-services

Two new initialization lists contain forms for enabling and disabling servers and services. (Initialization lists are explained in *Lisp Machine Manual*, p. 490.)

enable-services *Initialization list*

The forms on **enable-services** are run by **si:enable-services**. In addition, they are run automatically by **lisp-reinitialize** when a non-server Lisp machine is warm-booted or cold-booted.

disable-services *Initialization list*

The forms on **disable-services** are run by **si:disable-services**. In addition, they are run automatically by **before-cold** when you use **disk-save**.

8.2.7 Controlling servers

You can now designate which machines at a site are to provide services to other machines. *Server machines* are those Lisp Machines providing services, like file serving, tape serving, or hardcopy serving, to other Lisp Machines. (The kinds of services are defined by the site options, see *Software Installation Guide*.)

:server-lisp-machines is a new site option. Its value is a list of strings, the full names of machines that are servers.

Server machines do not automatically enable their servers when they are booted. This is to prevent premature creation of servers before the machine has been completely initialized (just seconds after being booted) or while someone is updating it (loading patches for example). Someone or something has to turn on the services:

```
(si:enable-services)
```

This can be done manually (after loading patches for example) or as the last thing in the server's init file. A note below the herald for the server machine reminds you that services are not enabled automatically.

All Lisp Machines can provide services. The distinction provided by **:server-lisp-machines** is that server machines are ones likely to be subject to continual requests for services whereas nonserver machines are likely to be asked only infrequently to provide service. The function **si:enable-services** runs whenever a nonserver Lisp Machine boots.

8.2.8 Enabling services

Two new functions control when services are available. You would not normally need to invoke these functions manually; the boot process takes care of the same functionality. Booting enables services for any machine unless it is a server Lisp Machine (see section 8.2.7, page 116.) Use these functions instead of **chaos:chaos-servers-enabled**.

si:enable-services
si:enable-services runs the **enable-services** initialization list (see section 8.2.6, p. 116).

si:disable-services
si:disable-services runs the **disable-services** initialization list (see section 8.2.6, p. 116).

8.2.9 New keyword for :esc-f-arg-alist option

The **:esc-f-arg-alist** site option now accepts the keyword **:all-lisp-machines**. This specifies all of the Lisp Machines in the host table as opposed to **:local-lisp-machines**, which specifies only those at your site. (See also **chaos:finger-all-lispms**, p. 68.)

8.2.10 New variable: supdup:*chaos-arpa-contact-name*

supdup:*chaos-arpa-contact-name* *Variable*
This variable specifies the type of server for a Chaosnet to Arpanet connection. It defaults to **tcp**. For connecting to a machine that still uses the NCP protocol, change the value of this variable to **arpa**. Only the name is different; the TELNET and SUPDUP protocols are the same.

8.2.11 Machine characteristics available to finger server

The finger server now returns whether a machine has a color monitor or a tape drive. It is not necessary to edit this information manually into the lmlocs file, if you were doing that previously.

8.3 Improvements

8.3.1 System shutdown initialization list

When **disk-save** runs, the following things now happen, in this order.

1. **before-cold** initializations get run.
2. **(logout)**
3. **system-shutdown** initializations get run.
4. The window and process systems are explicitly shut down.
5. The microcoded **sys:%disk-save** is called, which performs the saving.

The entries on the **system-shutdown** list should all be for subsystems that are required for almost anything else to run. Currently there are entries for the Chaosnet NCP, the TIME system, and the Lisp Machine file system. User programs should add themselves to the **before-cold** list rather than to **system-shutdown**.

This solves the problem that the Chaosnet would get shut down and its area unwired before the **cc** symbols were loaded using the Chaosnet, causing the machine to halt if too many page faults occurred.

8.3.2 Timeouts changed on file jobs

A file job consists of one control connection and a number of data connections. The default lifetimes of the connections for a file job have been changed:

Control connections	30 minutes (ITS); 2 hours (all others)
First data connection	12 minutes
Additional data connections	3 minutes

When a control connection times out, you must supply a password the next time you attempt to use the host (for hosts that require passwords).

:file-control-life-table is a new site option for specifying timeouts for control connections for various types of host systems. Its value is a list specifying the interval for each relevant host type. For example:

```
(:file-control-life-table
 '(("scrc-tenex" 108000.)                    ;30 minutes.
   (:its 108000.)                            ;30 minutes.
   (otherwise 432000.)))                     ;120 minutes.
```

Value	*Meaning of the* **cadr** *of the alist*
n	Timeout interval in 60ths of a second
nil, no entry	Control connection never times out

Valid host types:

:its, :lispm, :multics, :tenex, :tops-20, :unix, :vms

8.4 Notes

8.4.1 Appending to Lisp Machine file system dump tapes using TOPS-20 tape server

Attempting to append to the end of backup tapes when using the TOPS-20 tape server used to create unreadable tapes. This has been fixed; it now works to append to the ends of dump tapes, on all systems.

8.4.2 Bug in TOPS-20 Chaosnet NCP

The following problem has been discovered. Each TOPS-20 site should patch its monitor accordingly.

> The MOVEI T1,177777 at CHART1-1 should be HRRZ T1,CHAACK(CONN). As it is now, packets with a number greater than 100000 will never get retransmitted. As soon as one of them gets lost, the connection wedges. Since there is a timeout in SNPKBT, this also means that the connection will grab all of memory in its attempts to keep sending over the window size.

Index

Symbolics, Inc.

NOTES

Symbolics, Inc.

Symbolics, Inc.

NOTES

Symbolics, Inc.

Symbolics, Inc.

NOTES

Symbolics, Inc.

symbolics ™

Release 4.0 Release Notes
#990095

Design: Schafer/La Casse
Typesetting: Cover — Litho Composition Co.
Printing: Henry Sawyer Co.