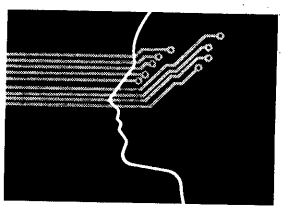
EXPLORER™ RELEASE 4.1
SOFTWARE RELEASE INFORMATION



READ
FIRST

2549844-0001•C

EXPLORER™ RELEASE 4.1
SOFTWARE RELEASE INFORMATION

MANUAL REVISION HISTORY

Explorer™ Release 4.1 Software Release Information (2549844-0001)

Original Issue ..... June 1987
Revision A ..... October 1987
Revision B ..... December 1987
Revision C ..... June 1988

The system-defined windows shown in this manual are examples of the software as this manual goes into production. Later changes in the software may cause the windows on your system to be different from those in the manual.

Texas Instruments Incorporated
ATTN: Data Systems Group, M/S 2151
P.O. Box 2909
Austin, Texas 78769-2909

Produced by the Publishing Center
Texas Instruments Incorporated
Data Systems Group
Austin, Texas

CONTENTS

EXPLORER RELEASE 4.1 SOFTWARE RELEASE INFORMATION

The following information describes the content of Explorer System Software Release 4.1. It outlines the goals for this release, the improvements made to the software, the new features that have been added, and issues of compatibility with the previous release.

1. Release 4.1 Goals

Release 4.1 was created to meet several major goals:

- Fix outstanding problems.
- Maintain compatibility with the previous release.
- Provide Extended Address Space (EAS) support.
- Add a remote tape facility.
- Add support for coexistence of monochrome and color screens on a color system.

2. Overview of Release 4.1

This release corresponds with the microExplorer™ 4.0 release. Since the microExplorer and Explorer releases were generated from the same source build, much of the source and object are identical. Fixes to any problems in these common areas will be the same. Since the microExplorer 4.0 release shipped first, the Explorer release was named 4.1 to designate a common source build with additional changes beyond what is in the microExplorer release.

Release 4.1 is being provided to customers in the form of bootable tape with a load band and microcode bands for Explorer I and Explorer II™. The microcode files for a specific machine must be installed for that machine. See the *Explorer System Software Installation Guide* for information on installing Release 4.1. Be sure to use the load distribution tape files from Release 4.1 when loading toolkits on Release 4.1. If you attempt to use the Release 3.2 load distribution tape files, an error will occur that says the streamer-tape system has not been loaded. During installation of Release 4.1, the streamer-tape files are restored by answering `"yes"` to the prompt `"Install microcode and help files on SYS-host <physical-host>?"`

Included with Release 4.1 are patches to the optional packages such as Explorer TCP/IP 3.0, NFS™ 3.0, TI Prolog 2.0, and DECnet™ 1.0. If any of these packages have been loaded in your band, a **load-patches** will automatically apply the patches to the optional package.

3. Kernel Changes

3.1 Microcode Changes

The Explorer II™ microcode has all LISP chip workarounds removed and therefore requires a LISP chip revision PG 4.2 or greater. Explorer II systems and processor boards shipped after November 1, 1987 have the new processors. If your system was shipped before November 1, 1987 and has not been updated, contact your local TI Field Service office for a free replacement processor. LISP chips earlier than revision 4.2 may not work properly with this release and should be replaced before going to Release 4.1.

3.2 Microcode Error Table

The microcode error table has been included in the microcode partition for Explorers I and II. In release 3.2, if the error table was not disk-saved in the load band, the error table had to be loaded from the sys-host at the beginning of every boot. Now, the error table will just be loaded from the microcode partition.

3.3 Extended Address Space (EAS)

The maximum virtual address space of the Explorer hardware is 128mb. Before EAS, if all 128mb had been allocated, the system could not allocate any more virtual memory, and the user would have to reboot. EAS gives the capability to extend the the virtual address space beyond the current 128mb hardware limitation. With EAS, we use this 128mb hardware supported address space to cover a single active environment.

With the temporal garbage collection (TGC) system and the Adaptive Training facility, we have a good foundation for an extension to eliminate the current address space limit by exporting objects from generation three (INACTIVE-3) to multiple inactive environments on disk, leaving necessary representative data structures behind. The size of the effective virtual address space is application dependent but can approach 1 gigabyte.

Without the EAS feature turned on, the system incrementally collects generations 0, 1, and 2. Any garbage collection of generation 3 is left to the batch functions invoked at the option of the user. Since objects which survive a generation 2 collection are promoted to generation 3, the size of generation 3 can grow. Indeed, it is possible for the size of generation 3 to grow above the size allowed for safe collection, even in batch mode. Since generation three is not collected incrementally, there is no automatic way to downgrade the activity classification of objects in generation 3.

With the EAS feature turned on, the system will incrementally collect generations 0, 1, 2, and 3. Inactive objects in generation three are exported to external environment storage and removed from the machine virtual address space. One can think of this external environment storage as a generation 4. Of course, any reference to an exported object will cause the object to be faulted back into the active environment. This fault process is very fast and generally entirely transparent. The collection of external environments is left to a batch facility, (**sys:gc-external**), invoked at the option of the user. An important distinction is that it will always be possible to safely perform a (**sys:gc-external**). However, this infrequent operation may be quite time-consuming if hundreds of megabytes are involved.

The following procedure must be taken to invoke EAS:

NOTE: EAS has been implemented only on Explorer II machines; the following procedure only has meaning for Explorer II's.

---

1. Configure your paging store to the desired size. The paging store size would normally be expected to be larger than 128mb in an EAS environment.
2. Place the form (**sys:eas-on**) in your login init file.
3. Boot and login with your login init file. This will place the system in a mode with **gc-on**, **training-on**, and **eas-on**. After this, just use your system.

NOTE: There is no (**sys:eas-off**) function and it is not possible to do a **disk-save** after the EAS feature has been turned on. If you want to do a **disk-save**, then just boot and login with a form (LOGIN 'name T). This will leave both Adaptive Training and EAS off.

4. Window System Changes

4.1 Menu Mouse Characters

Variables have been added to allow the user to control the mouse character that is displayed when selecting items from a menu.

Modifications to the *Explorer Window System* manual for release 4.1 — Add the following definitions in section 14.2.8 after **:mouse-standard-blinker.**

The next two variables can be modified from the User Profile utility. They control the shape of the mouse while it is on an exposed menu.

w:*menu-mouse-item-glyph* Variable

The value of this variable is a character object. When the mouse is over a mouse-sensitive item in a menu, the mouse glyph will change to this character in **fonts:mouse.**

w:*menu-mouse-no-item-glyph* Variable

The value of this variable is a character object. When the mouse is on a menu but not in a location that identifies a specific item, the mouse glyph will change to this character in **fonts:mouse.**

4.2 Multiple Screen Support

4.2.1 Introduction

Support for multiple Explorer screens has been added to the system. Explorer screens are at the top of the window hierarchy, and usually represent an individual piece of the display hardware. See pages 1-1 to 1-5 and 5-1 to 5-3 of the *Explorer Window System* manual for additional information about screens.

Multiple screen support is most useful on color Explorer II systems. Previously, **w:main-screen** was the only screen available to create and expose windows on. If color windows were desired, **main-screen** and all its inferiors had to be converted to color. Once **main-screen** was a color screen, no monochrome windows could be created. Now, with multiple screen support, **main-screen** should remain a monochrome screen. To create and expose color windows, simply create a color screen. The monochrome **main-screen** and the new color screen can coexist in the system, and you simply select which one you currently want exposed.

Details on how to create screens, switch between screens, and move windows from one screen to another will be discussed below. Though it is possible to move a monochrome window to a color screen, it is not possible to display a monochrome window on a color screen. This means that if a monochrome window is moved to a color screen, it will automatically be converted into a color window.

Though multiple screen support was added primarily for use on color systems, it is also available for use on non-color systems (those without a Color System Interface Board - CSIB). However, major benefits of using multiple monochrome screens have not been recognized. Note that, because of this, most of the discussion will be geared towards users of color systems.

The TI microExplorer system supports multiple screens, but in a different fashion. Thus, none of the functions or methods discussed below are supported on the microExplorer. Currently, color screens cannot be created on the microExplorer.

Multiple screen support addresses two concerns:

1. Color system performance.

In Release 3.2, the **w:convert-to-color** function converted **main-screen** to a color screen. From that point on, all bit-save arrays were 8-bit arrays instead of 1-bit arrays. This affects performance in two ways:

- Increased memory consumption, which leads to increased paging activity.
- Increased time to create windows and to have window's bit-save arrays bitblt'ed to the screen upon exposure.

Once **main-screen** was converted to color, there was no going back, and there was no method of creating a monochrome screen to take its place.

With Release 4.1, **convert-to-color** should no longer be used, though it is still supported for compatibility. Instead, if color windows are desired, a new screen should be created, which would be color. This way, most windows could remain as 1-bit windows, and fewer windows would have to be 8-bit windows. Thus, overall performance of the system should improve.

2. Allowing monochrome windows and color windows to coexist on a color system.

In Release 3.2, monochrome windows and color windows could not exist in the system at the same time. With multiple screen support, monochrome windows can be created on monochrome screens, and color windows on color screens. Only one of these screens can be exposed at a time on a single monitor, but now, all windows do not have to be color.

4.2.2 Using Multiple Screens

As mentioned earlier, the recommended way to make a color screen is by creating a new screen, and not by converting **main-screen**. The simplest way to do this is to use the function **w:create-color-screen**. This function will create a color screen and expose it. Initially, it will contain a Lisp Listener. The TERM CTRL-S key sequence or the Select Screen option of the system menu can be used to get back to the monochrome **main-screen**.

Each screen knows only about its own windows. If the same Explorer system application/utility (Program, Debug Tool) exists on both screens, doing a SYSTEM <keystroke> will not take you to the application on the other screen. For example, if one Zmacs editor exists on the monochrome screen and one on the color screen, and you do a SYSTEM E while on the monochrome screen, the monochrome screen's Zmacs will be selected. If SYSTEM E is performed again, a beep will occur. To get to the color Zmacs, you must first select (switch to) the color screen and then do a SYSTEM E. Also, a SYSTEM CTRL-E creates another Zmacs editor on the screen you are currently on.

In some situations, it may not be desirable, and may be impossible, to have two copies of an application existing on the same system. Thus, if both a monochrome and a color screen exist, a choice must be made. The application can be invoked and used on either the monochrome screen or the color screen. It is possible to move windows from one screen to another, but it's probably not something that should occur often. If monochrome windows are moved to a color screen, they get converted to color as they are moved. If color windows are moved to a monochrome screen, one of two things can happen. If **w:*convert-color-sheet-to-monochrome*** is true, color windows will always get converted to monochrome. If this variable is **nil**, and it is not explicitly specified in a

function call to convert color windows to monochrome, an error will occur if trying to move color windows to monochrome screens. See method **:set-superior** and function **w:move-sheet-to-another-screen**. Note that converting windows from monochrome to color takes a significant amount of time.

Note that if color windows do get converted to monochrome, some color information will most likely get lost in the conversion. That is, only two colors will survive the conversion. Speaking in terms of pixels on the screen, all pixels equal to the background color become zero, and all other pixels become one.

Some other things to keep in mind when using multiple screens:

a. Use a monochrome screen whenever possible — As mentioned previously, color windows with bit-save arrays affect performance of the system. Thus, the monochrome screen should be used as much as possible. Remember, many, if not all, of the Explorer system utilities, tools, and programs use bit-save arrays.

b. Load band portability — Support has been added to help ensure that load bands containing multiple screens, both monochrome and color, can be transported to other systems. Most importantly, if a load band which contains only color screens is moved to a system without color hardware, a new screen will be created and exposed. The **initial-lisp-listener** will get removed from **main-screen** and added as an inferior of the newly created screen. As it is moved from the color screen, it gets converted to a monochrome window. This conversion takes place while the system is booting, and may take a long time. The system may look as if it has stopped running (hung up). It may take 30–60 seconds for the conversion to take place and for the system to continue the booting process. If **main-screen** is left as a monochrome screen, as is recommended, then it will be chosen as the screen to expose at boot time, and no additional screens will need to be created.

c. Global variables — These variables are dependent upon which screen is currently exposed (active). Their contents change when switching from one screen to another.

   w:default-screen — Always equal to the screen instance which is currently active. Any references to **main-screen** in code should be replaced by **default-screen**.

   w:who-line-screen — Always equal to the currently active **who-line-screen**. This could be either a monochrome version or a color version.

   w:previously-selected-windows — Each screen keeps a copy of its previously selected windows. This global gets set to the screen's copy when the screen becomes active.

   w:cache-window — This instance is modified slightly depending on the type of screen currently active, monochrome or color.

   w:sprite-window — If a color screen is created, a color version of the sprite window is created. When the color screen becomes active, the color version of this is used. Otherwise, the monochrome (initial) version is used.

d. The global **w:all-the-screens** is a list of all the screens which have been created. The list should not be altered in any way.

4.2.3 Differences From Release 3.2

A new color map is being used with Release 4.1. The first 32 colors (system colors) are the same as before, but the remaining colors have changed. If the old color map is required, one can be created with the **w:create-color-map** function. This can be done by: **(w:create-color-map '(:type :color))**.

The sheet method **:set-superior** allows sheets to be moved from one screen to another. A conversion from monochrome to color or from color to monochrome may take place, if the type of the sheet is not the same as the type of the screen (see **w:*convert-color-sheet-to-monochrome***). Note that **:set-superior** may not refresh the sheet properly if a conversion is performed (see also **w:move-sheet-to-another-screen**).

Since an Explorer system containing color hardware (CSIB and color monitor) has the ability to be used as either a monochrome system or a color system, it follows that references to color system, color display, or color environment in the Explorer manuals typically mean color screen and references to monochrome system, monochrome display, or monochrome environment typically mean monochrome screen.

4.2.4 Obsolete Functions

w:convert-to-color — This function has been marked as obsolete. It can still be used, but is not recommended. Use **w:create-color-screen** or **w:make-a-screen** instead.

4.2.5 New Functions

w:make-a-screen — Makes a new screen, either monochrome or color.

w:create-color-screen — Creates a color screen.

w:move-sheet-to-another-screen — Moves a sheet from one screen to another. A conversion of the sheet may take place if the sheet is monochrome and the screen is color, or vice versa.

w:kbd-screen-redisplay-some — Redisplays only the screens in the list: **w:*screens-to-refresh***. Screens must be exposed to be refreshed.

w:system-menu-select-screen — Allows selecting a screen to expose from the menu of all available screens. This function is called when Select Screen is chosen from the Windows: column of the system menu.

w:swap-default-f-b — Swaps the values of **w:*current-foreground*** and **w:*current-background***. This can be used to reverse-video monochrome screens on color systems.

w:compress-array — A function similar to **bitblt**, except it copies 8-bit arrays to 1-bit arrays. It is not supported in microcode, and thus, is slower than **bitblt**.

4.2.6 New Methods

(sheet :sheet-legal-for-superior) — Determines if a sheet and its superior are either both color or both monochrome. If there is a mismatch, a conversion may take place.

(standard-screen :around :expose) — Used when switching from one screen to another. That is, when an **:expose** is sent to a screen, this method gets run. This method deexposes the currently active screen and exposes the new screen.

4.2.7 New Macros

w:color-sheet-p — More meaningful name for the **(color-system-p sheet)** macro. Both macros do the same thing. It is recommended to use the new macro.

4.2.8 New Instance Variables of Screen Flavor

screens-previously-selected-windows — Holds a copy of the screen's previously selected windows.

screen-descriptor — Holds a pointer to the **who-line-screen** which is active when the screen is active.

4.2.9 New Key Sequences

TERM CTRL-CLEAR-SCREEN — Executes the **w:kbd-screen-redisplay-some** to refresh exposed screens on the **w:*screens-to-refresh*** list. This is similar to TERM CLEAR-SCREEN which does screens on **w:all-the-screens**.

TERM CTRL-S — Useful for switching between color and monochrome screens.

4.2.10 New Item in Windows: Column of System Menu

Once an additional screen has been created, the Select Screen option becomes available from the system menu. Choosing this option will present a menu with a list of screens. Selecting a screen will expose it and make it the current **w:default-screen**.

4.2.11 Global Variables

w:*screens-to-refresh* — Contains screens to be refreshed by **TERM CTRL-CLEAR-SCREEN**. This list is not managed by the system. It is intended to be managed by the application developer.

w:*convert-color-sheet-to-monochrome* — If true, color windows will be converted to monochrome if an attempt is made to use a color window on a monochrome screen and when moving color windows to color screens.

4.2.12 Known Problems

Suggestions problem when moving windows — Moving a window which contains suggestions panes from one screen to another should be avoided, as this will cause a problem when exposing that window. This means that **initial-lisp-listener** should not be moved from **main-screen**.

It is advised not to invoke suggestions while using multiple screens since this may produce undesired results.

Removing all inferiors of a screen — This should be avoided. Doing so may lead to unrecoverable errors.

4.2.13 Problems Which May Occur

If an application does not work properly on another screen, the cause may be one of the following:

a. The application is referencing **w:main-screen** instead of **w:default-screen**.

b. Some array type may not match that which is required by a particular screen. Some examples are:

— Trying to indirect a 1-bit array to an 8-bit array.

— Trying to bitblt an 8-bit source array to a 1-bit destination array.

5. User Interface

5.1 System Access Menu

profile:system-access-menu is a new function which allows you to edit the SYSTEM key assignments. Items which appear on the system menu that have SYSTEM key assignments are editable by this function.

System menu and SYSTEM key assignments are made by using the function **w:modify-system-access-spec** and by modifying the **defsystem** of a subsystem. Existing code which calls the functions **tv:add-system-key** or **w:add-to-system-menu-column** will not be affected.

Using the function **w:modify-system-access-spec** allows you to assign SYSTEM keys to systems that may or may not be loaded. The user is prompted for whether he wants the system loaded when he tries to use a SYSTEM key or an item on the system menu to invoke an unloaded system which has been assigned in this manner.

The needed **defsystem** keywords are:

:name — The name to be displayed in system menu.

:documentation — The documentation used in the mouse line with the system menu and in SYSTEM HELP.

:default-menu-column — The column the system appears in by default. One of **:user-aids, :programs, :debug** or **:none.**

:default-system-key — The default SYSTEM key for this system or **nil.**

:instance-type — One of **:flavor, :window** or **:eval.** If your system uses a flavor call to instantiate itself, use the **:flavor** keyword. A window may also be used as the initiator of your system. If so, use the **:window** argument here. If you have a form that is to be evaluated to initiate your system, then use the **:eval** keyword.

:instance-finder — This argument depends on the **:instance-type.** When the SYSTEM key or system menu name is selected and instance type is:

:flavor — This argument is the name of the flavor.

:window — This argument is an instance of a window flavor.

:eval — This argument is the form to be evaluated.

:instance-creator — This argument determines what action will occur when a new instance is made via the SYSTEM CTRL-<assigned-key>. The same action is taken when a middle click is selected from the system menu. The values for this argument are:

T — If a new window or flavor instance as described in **:instance-finder** should be created.

NIL — Message is displayed notifying that a new instance cannot be created.

(form) — This form is evaluated when your subsystem is entered via pressing SYSTEM CTRL-<assigned key> or middle click is selected from the system menu.

If you are creating a new system and want to add it to the system menu or define a SYSTEM key for that system, use the keywords described above in your **defsystem** and the function **w:modify-system-access-spec**.

If there is no **defsystem** for your system, create one using a unique system-name for the system and the new keywords explained above. Once the **defsystem** is written, you need simply to replace the call or calls to **add-to-system-menu-column** and **add-system-key** with one call (**w:modify-system-access-spec** *system-name* **:assign-defaults**).

w:modify-system-access-spec (*system-name* &rest options) Function

This is the programmatic interface to the system access facility. If you want to add, change, or remove a system's access specification, use this function.

system-name specifies the system whose access spec is being modified. What is done to it is specified by the options. If no options are specified, this function simply makes sure that an access spec exists for *system-name*. The options are as follows:

:remove — Takes the system off the system menu, off the SYSTEM keys, and out of ***system-access-specs***. This option will override all others.

(**:assign-name** *name*) — Change the system's print-name to *name*.

(**:assign-column** *column*) — Change the system menu column in which the system appears to *column*, one of **:user-aids**, **:programs**, **:debug**, or **:none**.

(**:assign-key** *key*) — Change the system's SYSTEM key to *key*.

:assign-defaults — Assign the system's default print-name, the system's default system menu column, and its default SYSTEM key, if the key is not already taken.

w:*site-contents-pathname* Variable

The pathname of the file containing the systems available on **sys:site;**.

w:read-site-contents Function

Read contents file on **sys:site;**.

w:add-to-site-contents *(system-name)* Function

Add a system to contents file on **sys:site;**.

w:remove-from-site-contents *(system-name)* Function

Remove a system from contents file on **sys:site;**.

w:make-new-site-contents Function

Clear contents file on **sys:site;**.

w:make-site-contents Function

If no contents file on **sys:site;** exists, create one.

w:find-system-instance (*system-name* **create create-new-screen-p** &optional (**make-system-if-needed :ask**)) Function

Find or create an instance of the system named *system-name*. If this system does not exist, then it will be made first. An instance of it will then be found, or created if none exists yet, and selected as specified by the system's **defsystem**.

The system access menu provides a list of systems that are currently available. This menu will allow you to make these systems accessible from the system menu and/or the SYSTEM keys. The menu is invoked by entering: (**profile:system-access-menu**). The choices from the menu are:

Abort — Exit the system access menu, canceling any changes made on the menu.

Do It — Exit the system access menu, activating any changes made on the menu.

Search — Looks for all systems and updates the current list. It must first exit system access menu, so it will ask if you want to cancel any previous changes or if you want to activate them. It will then exit temporarily while it updates its list of systems, and then it will return to the menu with the updated list. It will find all systems that are either:

1. In the environment and are of **instance-type :flavor, :window,** or **:eval.**
2. Listed in the **sys:site;contents** file.

Modifying a System's Access Specification — The user can modify the name that is used for the system in the system menu and SYSTEM HELP, the column of the system menu in which the system appears, or the SYSTEM key that is assigned to the system by simply clicking on that item for that system. If the user clicks left, he is asked to input a new value; if he clicks right, the system's default-value is used.

Print Name — The system's print name is used on the system menu and by SYSTEM HELP. If you click left on a system's print name, a window will pop-up in which you can edit the name. Note that the name can be no more than 16 characters.

Menu Column — This specifies in which column of the system menu the system will appear. Clicking left on this brings up a menu of the possible choices: User-aids, Programs, Debug-tools, or None (you cannot add a system to the Windows column).

NOTE: If the system name is **nil**, the menu column field should not be modified.

System-key — This is the key that, when pressed following the SYSTEM key, will access this system. Clicking left on a system's key brings up a prompt for a new key.

5.2 New User Utility

The New User Utility has been removed from the system. Its function can be duplicated by creating a home directory and logging in as follows:

```
(LOGIN '<user-id>)
(FS:CREATE-DIRECTORY "lm:<user-id>;")
```

`<user-id>` is typically some short name specific to the user such as a first name or a last name.

6. Development Tools Changes

6.1 Glossary

The Glossary Utility has been moved to the PUBLIC directory. Its functionality has been replaced by Visidoc.

6.2 Band-Sizing Tools

A set of tools has been added to the system to facilitate the customizing and down sizing of a load band to include only the sub-systems required. The documentation of the band-sizing tools is included as Appendix A in this manual and was originally included in the microExplorer documentation.

6.3 Documenter System

The Documenter system is a set of functions that are intended to make understanding and documenting Lisp programs easier. These functions collect information from the current Lisp environment in virtual memory and display it in textual form. These functions are documented in Appendix B of this manual and were originally included in the microExplorer documentation.

6.4 Mail Summary Display

A mail variable to customize the mail summary display has been added and is documented as follows:

mail:*user-mail-summary-template* — This variable is a property list of alternating keyword-value pairs that determine the contents of the summary lines in the summary buffer and the order of items within each line. The keywords and their values are as follows:

:length — The possible values are **:lines** and **:chars**. When this keyword is set to **:lines**, the message size is shown by the number of lines; when this keywords is set to **:chars**, the message size is shown by the number of characters. The default value is **:chars**.

:from — The number of spaces to allow for the From: field. The default value is 30. It is 20 for microExplorer.

:subject —The number of spaces to allow for the Subject: field. The default value is 30. It is 25 for microExplorer.

:date — These values determine the contents of the Date: field. The permissible values are as follows:

:date-and-time — Day, month, year, and time of message origin.

:date — Day, month, and year of message origin.

:brief-date (the default value) — Day and month of message origin.

:keywords — The number of spaces to allow for the keywords field. The default value is 30. It is 17 for microExplorer.

6.5 DEFSYSTEM Options

Two new **defsystem** options, **:serial** and **:compile-load-modules**, have been added to provide for more control over when files are loaded during a **make-system**. The layout of the **sys::file-transformation** structure (used internally in **make-system**) has been changed to accommodate

the new **defsystem** options. This change requires recompiling any user defined functions that use accessors such as **sys::file-transformation-args**. A warning about "...compiled with a different version of macro **sys::file-transformation...**" will be seen when loading any XLD file that needs to be recompiled.

The **:serial** option documentation should be added on page 23-3 of the *Explorer Lisp Reference* manual before (**:not-in-disk-label**):

```
(:SERIAL {T | Nil})
```

If the value is true, this forces **make-system** to process transformations exactly in the order they are specified. Load and compile dependencies aren't required or used (but source level dependencies still apply via **:compile-load-init** transformations). If you say to `:compile-load foo`, it compiles files in module `foo` (if needed), and immediately loads files in module `foo` (if needed) instead of the normal behavior of trying to compile as much as possible in the entire system before loading. To get total sequential behavior, it would be necessary to have only one file per module. The default for the serial option is **nil**.

Insert the following at the end of the first paragraph on page 23-5 of the *Explorer Lisp Reference* manual:

The following is an example of the use of the **:serial** option:

```
(defsystem test-defsystem
    (:NAME "test-defsystem")
    (:PATHNAME-DEFAULT "lm:mk-sys-test;")
    (:MODULE foo "foo" :NEVER-SHIP)
    (:MODULE bar "bar")
    (:MODULE foo-sons ("foo-son1"
                       "foo-son2"))
    (:COMPILE-LOAD foo)
    (:COMPILE-LOAD bar)
    (:COMPILE-LOAD foo-sons (:fasload foo)))
```

The order of the execution of compile and fasload operations is as follows:

```
compile foo
compile bar
fasload foo   ;;Because (:fasload foo) was specified as a
              ;;dependency to (:compile-load foo-sons ...)
              ;;transformation
compile foo-son1
compile foo-son2
fasload bar
fasload foo-son1
fasload foo-son2
```

When the serial option is set to true, that is:

```
(defsystem test-defsystem
    (:NAME "test-defsystem")
    (:PATHNAME-DEFAULT "lm:mk-sys-test;")
    (:SERIAL t)
    (:MODULE foo "foo" :NEVER-SHIP)
    (:MODULE bar "bar")
    (:MODULE foo-sons ("foo-son1"
                       "foo-son2"))
    (:COMPILE-LOAD foo)
    (:COMPILE-LOAD bar)
    (:COMPILE-LOAD foo-sons (:fasload foo)))
```

Then the order of the execution of compile and fasload operations is as follows:

```
compile foo
fasload foo
compile bar
fasload bar
compile foo-son1    ;; The order of execution of compile and fasload
compile foo-son2    ;; operations within a module are done
                    ;; non-serially.
                    ;; If you need serial behavior within the files
                    ;; in a module, you need to specify one file
                    ;; per module.
fasload foo-son1
fasload foo-son2
```

The **:compile-load-modules** documentation should be added on page 23-8 of the *Explorer Lisp Reference* manual after (**:compile-load-init ...**)

```
(:COMPILE-LOAD-MODULES TARGET-FILE-SPEC . DEFAULT-OPTION)
```

This can be viewed as a combination of **:module**, **:compile-load-init**, and **:serial**. It is useful for defining systems involving many files of macro definitions that need to be processed in a certain order, using dependencies like **:compile-load-init**.

`target-file-spec` is a list whose elements are of the form **target-file-string** or (**target-file-string file-dependencies target-file-option**). **target-file-string** and **file-dependencies** can be a file-string or a list of file-strings. This specifies that each target file will be compiled when either its source has changed or the source or binary file of any of the dependency files has changed, and will then be immediately loaded if needed. (If the list of dependency files is empty, then the transformation is equivalent to **:compile-load** instead of **:compile-load-init**.)

The same options described previously for the **:module** clause can be specified by **default-options** for all the files, or overridden locally by **target-file-options**.

When using the **:compile-load-module** option the **:serial** option defaults to true and each file is treated as a separate module so that the files are processed in the exact order listed.

An example of the **:compile-load-modules**

```
(:compile-load-modules  ("lm:test;file1"
                         ("file-a" "lm:test;file1")
                         ("file-b" nil :package "USER")
                         ("file-c" ("file-a" "file-b") :never-ship-p t)
                         (("file-d" "file-e"))
                         (("file-f" "file-g" "file-h") "file-d")
                         (("file-x" "file-y") ("file-c"))
                        ) :package "FS")
```

This is equivalent to the following modules and transformations:

```
(:serial t)
(:module |LM:TEST;FILE1| "lm:test;file1" :package "FS")
(:module file-a "file-a" :package "FS")
(:module file-b "file-b" :package "USER")
(:module file-c "file-c" :never-ship-p t :package "FS")
(:module file-d "file-d" :package "FS")
(:module file-e "file-e" :package "FS")
(:module file-f "file-f" :package "FS")
(:module file-g "file-g" :package "FS")
(:module file-h "file-h" :package "FS")
(:module file-x "file-x" :package "FS")
(:module file-y "file-y" :package "FS")
(:compile-load |LM:TEST;FILE1|)
(:compile-load-init file-a (|LM:TEST;FILE1|))
(:compile-load file-b)
(:compile-load-init file-c (file-a file-b))
(:compile-load file-d)
(:compile-load file-e)
(:compile-load-init file-f (file-d))
(:compile-load-init file-g (file-d))
(:compile-load-init file-h (file-d))
(:compile-load-init file-x (file-c))
(:compile-load-init file-y (file-c))
```

On page 23-8 of the *Explorer Lisp Reference* manual, the second paragraph after **:compile-load-init**, the line "If the creation date of the source files defined by **add-dep** are newer ...." should be modified to say: "If the creation date of the source or binary files defined by **add-dep** are newer ...."

On page 23-15 of the *Explorer Lisp Reference* manual, the paragraph above the dark line, the sentence "Thus, the file or module to which the **:compile-load-init** transformation applies is compiled if it, or any of the source files it depends on, ..." should say, "Thus, the file or module to which the **:compile-load-init** transformation applies is compiled if it, or any of the source or binary files it depends on, ..."

7. I/O System and Device Support Changes

7.1 Remote Tape Capability

In Explorer Release 4.1, a remote tape capability has been added. This allows tapes to be accessed over the network. This feature only supports tape drives connected to Explorers; tape drives connected to other computers are not supported. The Explorer initiating the remote tape request is called the client. The Explorer connected to a tape drive that is handling the remote tape request is called the server. Both the tape client and the tape server must be running Explorer Release 4.1 or greater.

All of the tape formats supported on local tape drives are also supported as remote drives.

All of the features supported on an Explorer local tape are also supported via remote tape except for the following operations:

- Position Past Blocks
- Make Bootable Tape
- Restore Bootable Tape
- Verify Bootable Tape
- Load Distribution Tape

Remote tape activity, like local tape activity, is initiated from the Tape Backup window (SYSTEM B). To start a remote tape operation, select Prepare Remote Tape from the Tape Backup menu. You will be prompted for the tape format and the host name of the tape server. Prepare Remote Tape makes sure the unit is available and prevents access to that tape unit by other users until it is free. If the tape unit on the selected host is available, you will be presented with a menu of available tape operations. If the tape unit is in use, a message will be displayed indicating that it is in use and who is using it.

After the tape has been prepared, it can be used to back up, restore, and verify files, directories, and partitions.

After your remote tape operation is complete, you should do one of two things:

1. Select Unload from the Tape Backup menu. This logs you out from the tape server and makes the tape drive unavailable to others until the tape is physically replaced in the drive.

2. Select Logout from the Tape Backup menu. This logs you out from the tape server, but will allow other users to append data to the same tape.

8. Networking Changes

8.1 DECnet Toolkit

Changes to the Ethernet® code have eliminated the need for the NuBus™ package and will cause DECnet to go into the error handler during installation. Patches to DECnet which are included in the system source tape handle the necessary changes.

If you are installing DECnet from the load distribution tape you will need to enter the following form in your Lisp Listener before you begin:

```
(pkg-find-package 'nubus t)
```

8.2 Telnet VT100 Screen

The LED pane was removed from the screen. The VT100 and Telnet Commands pane has been moved to a pop-up menu which you can invoke with a middle-mouse click.

9. LX 3.0 Installation

In order to upgrade an Explorer LX™ system to Explorer Release 4.1 (or later), you must have Explorer LX 3.0. The LX 3.0 release will start shipping about one month after the Explorer Release 4.1. LX 3.0 also depends on TI System V 2.2.0 (or later). To upgrade an Explorer LX to Explorer Release 4.1, LX 3.0, and TI System V 2.2.0, follow the procedure in the *Explorer LX System Installation Guide* titled "Upgrading an Explorer LX to LX Release 3.0".

10. Ergonomics

In order to satisfy requirements of the European VDE safety regulation ZH1/618, the default video mode is white characters on a black background. This mode is recommended for extended periods of use to reduce any perception of flicker from the display. To change between video modes on a monochrome system, press TERM C.

To use a larger font that provides better contrast than the default font of the Explorer, use the form **(w:set-default-font 'fonts:medfntb)**. To change back to the original default font, use **(w:set-default-font fonts:cptfont)**. This command changes the default font in all windows.

The Explorer system allows the user to construct and select other fonts suited to specific applications or to meet special requirements. For example, certain international standards have specific requirements for character descenders and ascenders. Using the Font Editor, the MEDFNTB or CPTFONT fonts could be modified, renamed, and installed to meet these international standards. For detailed information, see the Font Editor section of the *Explorer Tools and Utilities* manual.

11. Compatibility with Previous Releases

There is very strong compatibility between Release 4.1 and Release 3.2. Applications that run on 3.2 should only have to be reloaded on 4.1 and should then operate as they did on 3.2.

12. Restrictions

12.1 Graphics Window System (GWIN)

On a color screen, raster objects and rasterized subpicture objects are very expensive in both terms of memory usage and file usage. It is recommended to limit the number of these objects in GWIN applications. See the explanation of **gwin:*current-cache-for-raster-objects*** variable in the *Explorer Tools and Utilities* manual.

On a color screen, the **rasterize-objects** and **make-sprite-from-objects** functions may not produce the desired color for objects that are raster or subpicture objects.

If a picture file created on a color screen contains raster objects or rasterized subpicture objects, and this picture file is used on a monochrome screen, the raster objects will not appear correctly.

Color printing is not supported for picture files. Pictures created on a color screen may be printed, but all colors are mapped to some gray stipple pattern when printed.

Drawing a closed polyline may not always work correctly. The result may be that the last line segment will extend beyond the specified closing point.

Drawing filled polylines or splines may not always work correctly. Holes (gaps or glitches) may appear in the interior of the object.

The **print-graphics** function discussed in the graphics section of the *Explorer Window System Reference* manual is in the USER package, not in the printer or GWIN packages. However, GWIN must be loaded to use it.

The **user:print-graphics** function can be used to print picture files to serial printers. This function cannot be used to print on parallel printers. The only way to print pictures on parallel printers is to use TERM Q in the Graphics Editor.

12.2 Graphics Editor (GED)

Paint objects are very expensive when on a color screen in both terms of memory usage and file usage. It is recommended to limit the number of these objects in any GED pictures. See the explanation of the **gwin:*current-cache-for-raster-objects*** variable in the *Explorer Tools and Utilities* manual.

When painting (drawing), it is strongly recommended that you complete the object being drawn in one paint session. That is, complete the entire painting and save it as one object. This is especially true when working on a color screen. Thus, the example in paragraph 10.4.7.2 in the *Explorer Tools and Utilities* manual would be more efficient if done as one object instead of three.

If **ged:*save-bits-for-buffers*** is **nil**, the Next/Previous buffer operation may cause the current buffer (picture) to be cleared from the screen. The picture can be restored by pressing the CLEAR SCREEN key or selecting REDRAW from the Pictures menu.

On a color screen, rasterizing a subpicture is very expensive in terms of file usage and memory usage. It is recommended that you NOT save subpictures as rasterized objects.

On a color screen, when you are changing the background color of the GED, paint (draw mode) objects may not appear in the color expected.

If a picture file created on a color screen contains paint (raster) objects or rasterized subpicture objects and this picture file is used on a monochrome screen, these objects will not appear correctly.

Since the background color of GED is 12% gray, any objects drawn with 12% gray color will not appear. This is also true of any picture files loaded which contain objects drawn with 12% gray color.

On a color screen, the recommended way to change the color of graphics objects is with Edit Parameters and Set Edge/Fill Color (CTRL-C or from STATUS). The Color Map Editor is not the desired way to change colors in GED. GED does not save color maps with picture files. Thus, the next time the picture is loaded, the objects would appear in a different color if the color map has changed.

Painting with 12% gray cannot be used to erase areas of a picture. This is because Paint makes use of the transparency ALU which causes destination pixels to be unmodified if the source pixels are the same as the background color of the window.

If a picture is being created on a color screen, and is intended to be displayed on a monochrome screen, some thought should be put into selecting appropriate colors and ALU's. Some colors will not show up at all on a monochrome screen.

On color screens, while dragging (Drag Copy/Move) a paint object or a subpicture, the colors may not appear correct. The object will return to its proper color once the Drag Copy/Move operation is completed.

On a color screen, if you change the background color of the GED, paint (draw mode) objects may not appear in the color expected.

Color printing is not supported for picture files. Pictures created on a color screen may be printed, but all colors are mapped to some gray stipple pattern when printed.

Panning may sometimes distort the appearance of the picture. Pressing the CLEAR SCREEN key or selecting REDRAW from the Pictures menu will eliminate the distortion.

To return to GED after a background stream window has come up on top of GED, just click the mouse outside of that window.

Drawing a closed polyline may sometimes not work correctly. The result may be that the last line segment will extend beyond the specified closing point.

Drawing filled polylines or splines may not work correctly. Holes (gaps or glitches) may appear in the interior of the object.

When in paint mode in GED, you must turn off painting before using the system menu. Otherwise, a paint stroke will extend from the original mouse location to the mouse location after the system menu operation is complete.

When in paint mode, painting should be avoided outside of the GED window. Nothing unusual happens at the time, but when you try to save the painting, a bitblt error may occur. This situation can occur when painting with a wide paint brush and part of the width of a brush stroke runs off the top or left edges of the screen, even though some of the stroke is always visible. If this problem does occur, try restarting the paint process by mouse clicking left twice.

Objects in GED with sides smaller than their edge thickness may draw themselves in a distorted manner.

When writing picture files or saving status, presentation, or subpicture files, be sure to enter a valid directory pathname. If an invalid (non-existent) directory is specified, a GED background stream window will appear. To return to GED from the background window, just mouse click outside of that window. Note that the default pathname shown when saving a file may not be an existing directory.

Since the **user:print-graphics** function is called when you press CTRL-P or select Print from the Pictures submenu, the only way to print pictures from GED on parallel printers is to use TERM Q. CTRL-P and selecting Print will work for printing to serial printers.

12.3 Compiler

The way that **format** calls are now optimized can sometimes have the undesired effect of starting to write the first part of the message before all of the format arguments have been evaluated. In cases where this is a problem, a **dont-optimize** form can be wrapped around the **format** call to prevent the optimization.

When compiling a Zmacs buffer, errors reported by the reader may be labeled with the name of the section preceding the one that really has the error.

In certain contexts, the compiler ignores **notinline** declarations for local functions defined by **flet** or **labels**.

If a process overflows the PDL and you press RESUME to increase the PDL and continue, you may then get an error for a throw tag not found (the tag is a locative) if there is a non-local **return-from** that wants to exit a BLOCK that was entered before the PDL was thrown. If this should occur, just press ABORT and re-run the program; since the PDL is now larger it should run correctly the second time. (This is a problem with both compiled and interpreted code.)

12.4 Kernel

The evaluator does not permit local unspecial declarations to shadow a global special declaration.

macroexpand does not expand those Common Lisp macros which the Explorer actually implements as special forms. The following technique can be used instead:

```
(DEFUN REALLY-EXPAND (FORM &OPTIONAL ENVIRONMENT)
 (IF (ATOM FORM)
    FORM
  (LET ((EXPANDER (MACRO-FUNCTION (CAR FORM))))
    (IF EXPANDER
        (REALLY-EXPAND (FUNCALL EXPANDER FORM ENVIRONMENT)
                       ENVIRONMENT)
        FORM))))
```

Flavor methods cannot be traced when the flavor instance has a large number of instance variables; you get "Stack frame larger than 256 words" from **sys:apply-lambda**. This can be worked around by setting **compiler:compile-encapsulations-flag** to true before calling **trace** so that the method is compiled instead of interpreted.

12.5 Date and Time Initialization

Because of the date and time handling changes made in the system, the first time a stand-alone user boots the load band he/she will be prompted to enter the date and time. Certain initializations will then be done to registers on the Real Time Clock chip, and the user should not need to enter the time information on subsequent boots.

NOTE: If a networked user is unable to reach a time-server during the initial boot of this band, he/she will also have to enter the date and time.

12.6 Fonts

A font which does not have all of its characters defined will cause errors for some keystrokes, and if a partial font is used as the default font, it will cause errors when instantiating some types of windows. The following fonts are not fully defined:

cmdunh cmr18 cmb8 cmold cmr10 icons mouse ti-logo search 43vxms

The font named **medfntb** is now identical to the font named **medfnb**. The Explorer will be phasing out the font named **medfnb** since it does not follow the standard naming convention for fonts. It is recommended that users modify any code and file attribute lists that refer to font **medfnb** to now refer to **medfntb**.

12.7 Profile

In the File System Variables section, if you fill in an invalid hostname for the **user-login-machine**, you will enter the error handler. If you do, press META-CTRL-ABORT, point at the entry again, and enter a valid host name.

If you enter the error handler when in Profile, it appears in a typeout window. When you exit the error handler, you may hang in Output Hold state. If so, press META-CTRL-ABORT to proceed.

12.8 Suggestions

In Zmacs Suggestions menus, the first time each menu is displayed and the mouse is moved from item to item, it will take up to 3 seconds for the mouse blinker to box the item. The problem is in the time required to calculate the mouse documentation string (specifically, calculating the keystroke for the current item). Subsequent times the mouse is positioned over the same item the time to display it is negligible since the documentation string is cached once it is calculated. Note that this problem is specific to Zmacs menus for Suggestions.

Within Suggestions' Menu Tools, the Menu Search option does not search for a string through the new menus that are created by the user. Also, the Find Functions for the Menu option does not display the commands in the Suggestions menu if you are in the Lisp Listener.

12.9 verify-cfg-boot

When running the **sys:verify-cfg-boot** function, you may see the error, "the function **nil** is undefined". A known cause for this is when the default boot unit kept in non-volatile RAM, NVRAM, is **nil**. To verify that this is the problem, enter (**sys:nvram-default-unit**). If it returns **nil**, you should setup NVRAM using the **sys:change-nvram** function passing it the **:load-unit** keyword and the disk number of your boot disk.

12.10 Visidoc

The text and images in the online manuals for Visidoc do not exactly match the printed manuals. Known problems are:

- Screen dumps, visible in the hardcopy manuals, are replaced with text that says "See page X-XX in the manual."
- Sometimes items that are supposed to begin numbering with 1 instead begin numbering with the current chapter number.
- Charts with data in tabular positions do not always line up correctly.
- Sometimes lozenge characters are printed separately on the next line.
- Whenever super- and sub-script characters appear, they show up on the following line.
- When multiple definitions occur at the same place in the manual (for example, see "FIRST", "SECOND", etc.), the text will be duplicated in your Visidoc buffer.
- Occasionally, the first time a machine tries to access a Visidoc server it fails and enters the error handler. If a server is actually active on the network, pressing RESUME will usually cause it to immediately contact the active server.

13. Documentation Changes

13.1 General

Release 4.1 provides revisions and change packages of some of the system software manuals. The following chart shows the system software manuals and how they have changed.

Title	Change
Explorer System Release Software Installation	Revision

13.2 Explorer Lisp Reference Manual

Following are some corrections to the *Explorer Lisp Reference* manual:

On page 3-17, in the example of **integer-decode-float**, the last value returned should be 1 instead of 1.0.

On page 10-6, under both the **:copier** and **:predicate** options, delete the phrase: "(on the Explorer system, a string is also allowed)".

On page 10-11, the description of the **:alterant** option is incorrect. Beginning with the second sentence, the rest of the paragraph should say: This option accepts one argument, which should be a symbol. If the argument is not specified or is **nil**, then no alterant macro is created. If this option is supplied without arguments, then the name of the alterant macro is a concatenation of ALTER- and the structure name.

On page 16-20, the paragraph in the middle of the page that begins "The environment argument ..." should be replaced with "The optional environment argument is currently ignored and should not be used."

On page 18-5, under "&body", delete everything after the first paragraph since the alternate syntax is not currently supported.

On page 18-14, under "parse-body", delete the last sentence "The &body ...".

On page 19-27, under flavor "**sys:print-readably-mixin**", add the following second sentence: "For this to work properly, all of the relevant instance variables must be declared as inittable or settable."

On page 19-27, section 19.11, delete the second sentence and add "See" as the first word of the following sentence.

On page 19-28, under operation "**:clear-hash**", delete the last sentence: "These functions ... operations."

Of the functions that are marked with "[c]" as being standard Common Lisp, several are actually additions to the language that have been proposed since the publication of the 1984 book *Common Lisp the Language*, and it is not clear at this time whether they will eventually become officially adopted. These are:

Function	Page
copy	9-3
delete-setf-method	2-22
hash-table-rehash-size	11-2
hash-table-rehash-threshold	11-2
hash-table-size	11-2
hash-table-test	11-2
parse-body	18-14
row-major-aref	7-9
signed-ldb	3-24
type-specifier-p	12-9
uncompile	21-2
user-name	*Tools and Utilities* 25-19
xor	14-21

13.3 Explorer Networking Reference Manual

The documentation in the *Explorer Networking Reference* manual must be revised as follows:

Paragraph 3.6 on page 3-16 describing the Name protocol is wrong. The finger function is a high level interface into the ARPANET Name protocol.

The following information should be added to paragraph 3.7 on page 3-16:

The **net:host-time** function and **net:host-uptime** function have been made generic to run over multiple protocols. The TI namespace must be changed in the following manner:

- Host objects with Chaos addresses should add the following services, if supported by that host:

```
(:TIME :CHAOS-SIMPLE :TIME-SIMPLE)(:UPTIME :CHAOS-SIMPLE :UPTIME-SIMPLE)
```

- Host objects with IP addresses should add the following services, if supported by the host:

```
(:TIME :UDP :TIME-SIMPLE-MSB)
(:TIME :TCP-SIMPLE :TIME-SIMPLE-MSB)  ;only if udp time-simple-msb
                                      ;support not available
```

- TI Explorers support all of the above mentioned services. Therefore, (**:time :udp :time-simple-msb**) should be used on an Explorer rather than (**:time :tcp-simple :time-simple-msb**) because it is a lower overhead service implementation. The Wollongong Group's WIN/TCP™ Release 3.0 does not support (**:time :udp:time-simple-msb**), but does support (**:time :tcp-simple:time-simple-msb**).

Paragraph 3.9 on page 3-17 should indicate that the finger function is an interface to both a Lisp machine version of the ARPANET Name protocol for Chaosnet hosts and the actual ARPANET Name protocol for TCP/IP hosts. This establishes a full connection with retransmission.

The TERM F key sequence is an interface to a Lisp machine version of the ARPANET Name protocol for Chaosnet hosts and for TCP/IP hosts. The TERM F key sequence uses a simple transaction instead of a stream connection.

The following information should be added to paragraph 3.9.1 on page 3-17:

The **net:finger** function, **net:who** is function, and TERM F "finger" functionality has been made generic to run over multiple protocols. The TI namespace must be changed in the following manner:

- Host objects with Chaos addresses should add the following services, if supported by that host:

```
(:SHOW-USERS :CHAOS-STREAM :NAME)(:LISPM-FINGER :CHAOS-SIMPLE :LISPM-FINGER)
```

- Host objects with IP addresses should add the following services, if supported by that host:

```
(:SHOW-USERS :TCP-STREAM :ASCII-NAME)(:SHOW-USERS :UDP-STREAM
:ASCII-NAME)(:LISPM-FINGER :UDP :LISPM-FINGER)
```

- TI Explorers support all of the above-mentioned services.

In the first paragraph on both page 3-17 and 3-18, the function, **chaos:find-hosts-or-lisprns-logged-in-as-user**, should be **net:find-hosts-or-lispms-logged-in-as-user**.

On page 4-7, the paragraph labeled **:ip-addr-subnet-bits** should be deleted. This attribute is no longer supported. The note following this paragraph regarding (**:gateway :ip :ip-gateway**) should be deleted as well.

The first paragraph at the top of page 6-13 should read:

The **:connection-possible-p-function** argument is a function or function name that returns **t** if it is possible to get a connection from one host to another on this medium. If in doubt, return **t**. The arguments to the **:connection-possible-p** function are *host-a*, *host-b*, and *logical-contact-name*.

The description of the **net:connection-possible-p** function on page 6-14 should read:

net:connection-possible-p *medium host-a* Function
&optional (*host-b* **sys:local-host**) *logical-contact-name*

This predicate returns **t** if it is theoretically possible to create a connection from *host-b* to *host-a* on at least one implementation of *medium*.

The *medium* argument is a keyword (or string) that represents the medium or the actual medium object.

The *host-a* argument is a string representing the host name of the host to be contacted, or the actual host object itself.

The *host-b* argument is a string representing the host name of the host originating the connection, or the actual host object itself. The default binding of this parameter is **net:local-host.**

The *logical-contact-name* argument can be either a string representing the contact name of the server process being solicited, or the actual logical-contact-name object itself.

On page 7-8, the Peek Network Mode Ethernet® Selection Screen now looks like the following:

Type	Slot	Subnet	#-In	#-Out	Aborted	Lost	FCS-Error	Timeout	Too-Big
NuBus	F0	0	49469	748	0	0	0	0	0

Ethernet Meters

40	NuBus F0 Total Number of packets received
35	NuBus F0 Total Number of packets transmitted
0	NuBus F0 Total Number of packets too big to receive
0	NuBus F0 Total Number of packets received with CRC errors
0	NuBus F0 Total Number of packets lost by hardware
1005	NuBus F0 Number of Ethernet broadcast packets received
0	NuBus F0 Collisions on ether
0	NuBus F0 Number of NuBus errors encountered
0	NuBus F0 Number of frames actually read with errors

On page 7-10, the Network Operations menu now looks like this:

Network Operations

Create Network Controllers
Reset All Network Controllers
Reset One Network Controller
Reset All Network Meters
Clear Address Translations
Reset Routing
Table Disable Enable
Reset Displays Diagnostics
Quit

The changed items are described as follows. Note that these items have been generalized for all types of controllers, not just Ethernet.

Create Network Controllers — Creates new data objects for all network controllers on the local host. Each network controller board on the system must have a related software object; the NuBus is scanned for network controllers and the correct object for that controller is automatically created. This is done at boot time.

Reset All Network Controllers -- Resets all network controllers and their associated objects (hardware and software reset). Higher level networking operations are not disrupted by this action.

Reset One Network Controller — Same as Reset All Network Controllers if only one controller exists; if more exist, a menu of the controllers is presented so that one can be selected for the reset.

Reset All Network Meters — Resets all network meters to zero. This encompasses both controller and global network meters.

Clear Address Translations — Deletes all entries cached in the logical to physical address translation tables.

On page 7-12, the Network Display menu now looks like this:

```
Network Displays

Print Chaos Address Translations
Print IP Address Translations
Print Controller Status
Print Controller Statistics
Reset Controller Statistics
Print Address Translations
Print STS Why
Print Recent Headers State
Print Routing Table
Print Pkts from Ether Diagnostics
Clear Screen
Quit
```

The changed items are described as follows:

Print Controller Status — Displays the state of the network controller. This display is hardware dependent, usually the slot, physical address, and internal state is displayed.

Print Controller Statistics — Displays the network meters internal to the network controller.

Reset Controller Statistics — Resets the network meters internal to the network controller.

On page 7-13, the Network Diagnostics menu now looks like this:

```
Network Diagnostics

Monitor Controller Link
Execute Reflectometer Test
Execute Controller Selftests
Dump Controller Memory
Show Routing Path
Show Routing Table
Displays
Clear Screen
Quit
```

The changed items are described as follows:

Monitor Controller Link — Intercepts all frames from the controller's link and displays their contents. Clicking on this item turns off the controller's receiver, so all network communications over this controller are disabled while Monitor Controller Link is running.

Execute Reflectometer Test — Runs a Time Domain Reflectometer test on the network controller board and prints the results.

Execute Controller Selftests — Runs all selftests supported by the controller hardware, such as memory tests, chip diagnostics, etc. A message is printed to indicate whether the hardware passed or failed the selftests.

Dump Controller Memory — Dumps the internal memory buffers of the controller hardware. The format is dependent on the type of controller hardware.

On page 7-15, the Network Controller Configuration menu is no longer supported since controllers are now configured automatically.

13.4 Introduction to the Explorer System

Section 1.5.1 on New User Utility has been replaced with the following:

To login to the system, you need to create a user directory and then login. Do the following:

```
(login '<user-id>)
(fs:create-directory "lm:<user-id>;")
```

<user-id> is typically some short name specific to the user such as first name or last name. To logout, just enter (logout). The next time you wish to login, just re-enter (login '<user-id>).

14. Public Directory

14.1 Screen Saver Utility

When the monitor is left unattended for long periods of time, e.g., overnight, it is best to turn down the brightness so the phosphorus on the screen doesn't burn needlessly. The screen saver utility is an automatic tool which also helps to preserve the phosphorus. This feature makes the screen go black if there has not been any keyboard or mouse interaction after a designated period of time. As an additional feature, while the screen saver is awaiting one of these two events, a separate process is started which can be programmed to do some user specified task; the default is to draw the QIX pattern. When the screen saver utility is terminated it automatically kills this associated process.

14.2 Color Enhancements

The **sys:public.color-enhancements** directory was added. It contains code which implements features that are not built into the released window system.

15. Considerations for Future Releases

Zetalisp support will become an optional sub-system rather than part of the basic load band. Therefore, when writing Common Lisp programs it is best to avoid using functions from the ZLC package so that you won't depend on having the Zetalisp sub-system loaded.

Beware of writing code that depends on the fact that **self** and **self-mapping-table** are bound as special variables. In future releases, these may become local variables of the method.

16. Customer Interface to TI

16.1 Problem Reporting

If you experience problems with either hardware or software, please report those problems as soon as possible. A detailed description on how to submit bug reports is documented in the file **sys:help;how-to-report-bugs.text** and summarized as follows:

There are several ways to create online bug reports from an Explorer.

- If the bug causes the machine to enter the error handler, you can press CTRL-M while in the error handler.

- You can use the function (bug) from any Lisp Listener.
- You can use the Zmacs command META-X Bug.

With any of these means, you are presented with a Zmacs buffer in which you can describe your problem. The system captures your hardware and software configuration information for you. When you finish filling in the form, you can use one of these means to get the report to Texas Instruments:

- Mail the bug report by filling in the address in the TO and SUBJECT fields at the top of the form and pressing the END key. If you have access to ARPANET mail (or CSNET mail), you can mail the form directly to Texas Instruments at EXPBUG@CSC.TI.COM. Otherwise, mail it to a local bug report mailbox. The EXPBUG mailing list is read by Texas Instruments only.
- Print a copy of the form using the Zmacs META-X Print Buffer command and mail it to Texas Instruments at

  EXPLORER BUG REPORTS
  C/O Explorer Project Manager
  M/S 2201
  Texas Instruments
  P.O. Box 2909
  Austin, Texas 78769-2909

- Write the form to a file using the Zmacs command CTRL-X CTRL-W and specifying a file name, print the report, and send it to Texas Instruments.

If you encounter a critical problem that needs immediate attention, please contact the TI Explorer Support Center for help in the United States. You will need to supply your Technical Support ID when making the call. If you have a new system and you don't yet have a support ID, call (512) 250-6179. You will need to supply the system serial number for your machine. The serial number can be found in the lower left corner of the inside front door of your system chassis, and on the outside of the READ ME FIRST envelope or OPEN FIRST BOX that came with your Explorer.

International customers with questions or problems should contact the Texas Instruments organization in their local country. The local TI organization is able to provide hardware service and also software support. In addition, the local software support organization will be able to provide any patches available for customers.

16.2 Hardware Service

If your Explorer hardware needs service, you can reach Texas Instruments Field Service at (800) 572-3300. You will need to supply your system serial number when making this call.

16.3 Explorer Mailing Lists

There are two public mailing lists on the ARPANET that anyone with ARPANET access can add themselves to:

- Info-TI-Explorer@SUMEX-aim.stanford.edu — To add yourself to this list, send a message to Info-TI-Explorer-Request@SUMEX-aim.stanford.edu. This list is intended for discussion of general Explorer issues.
- Bug-Explorer@SUMEX-aim.stanford.edu — To add yourself to this list, send a message to Bug-Explorer-Request@SUMEX-aim.stanford.edu. This list is intended for reporting bugs or fixes for problems to the Explorer user community.

The above two lists are supported by Explorer users and not by Texas Instruments. However, by sharing experiences with other users, Explorer users will be more productive. Texas Instruments personnel do read messages on these lists.

16.4 Explorer User's Group — NEXUS

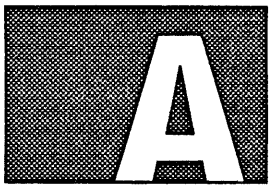
The National Explorer Users' Society (NEXUS) was formed in 1986, with an initial meeting at the AAAI meeting in Philadelphia. For further information, please contact one of the following:

Rich Acuff
acuff@sumex-aim.stanford.edu
Stanford University
Medical School Office Building
Stanford, CA 94305

Glenda McKinney
nexus@dsg.csc.ti.com
M/S 2201
Texas Instruments
P.O. Box 2909
Austin, Texas 78769-2909

16.5 Closed Problem Reports

The file **sys:help;closed-bug-reports4-1.text** lists bug reports closed in this release. The one line entries are sorted by bug number and include the original bug number plus the name of the system which fixed the problem.



BAND-SIZING TOOLS

Introduction

A.1 This section describes how to use the tools supplied with Release 4.0 of the microExplorer and with Release 4.1 of the Explorer for reducing the size of load bands (partition-files). Usually a load band is built by booting a starting band, using **load** or **make-system** to add additional capabilities, and then performing a **disk-save** to write the augmented band to disk. You can also use these tools to remove capabilities from a band before disk saving.

Sequence of Operations

A.2 To build a custom load band, perform the generalized steps given below. You can omit steps that are not applicable; however, for best results perform the steps in the order shown. For information about the functions and procedures required to perform these generalized steps, see the rest of this appendix.

1. Cold boot the band that will be used as the starting point.
2. Use the **load-patches** function to make sure all current patches are loaded.
3. Use **make-system** or **load** to load the features that are to be added to the band.
4. Load the band shrinking tools by executing the following form:

   ```
   (make-system 'shrink-tools)
   ```

   This will load the three functions **sys:delete-system**, **sys:band-cleaner**, and **sys:tree-shake**. Alternatively, these can be loaded individually from the SYS:BAND-TOOLS; directory by calling **load** on the file with the same name as the function.
5. Use the **sys:delete-system** function to undefine those features that are to be removed from the band (described later in this section).
6. If there are UCL-based applications in which certain commands are not needed, use the UCL command editor to delete the unneeded commands. Be sure to kill any windows that were created while deleting the unneeded commands.
7. Use the **sys:band-cleaner** function to selectively delete unneeded debugging and documentation information (described later in this section).
8. Perform the following functions:

   ```
   (sys:delete-system 'sys:band-cleaner)
   (sys:delete-system 'sys:delete-system)
   ```
9. Run the **sys:tree-shake** function to eliminate symbols that are no longer referenced (described later in this section).

10. Use the function **gc-and-disk-save** to save the band.

11. Boot the new band, and test it for its intended purpose. At this point you may discover items that have been deleted but which are needed, or items that you do not need but which were not deleted. If so, you must start again at step 1, modifying the options in accordance with what you have learned. If functions or variables are still present, but you think they are not being referenced, the cross-reference tools of the Documenter system or the **who-calls** function may be helpful to find where they are being referenced. As a last resort, you can use **sys:invert-tree** to find how objects are referenced (to find why data structures were not garbage-collected or to find references to functions from data structures).

12. Once you are satisfied that the contents of the band are correct, you can optimize the band by rebooting, performing load band training, and then saving the trained band. See the functions **sys:start-training-session** and **sys:end-training-session** in the *Explorer Lisp Reference* manual.

The sys:band-cleaner Function

A.3 The band cleaner function is defined as follows:

sys:band-cleaner &key :unused-pathnames :cdr-code-plists Function
:previous-defs :debug-structures :doc-strings :zmacs-properties
:compiler-properties :arglists-and-local-maps :pathname-properties
:verbose

The **sys:band-cleaner** function can be run to selectively delete certain data that is either no longer needed or that is used only for documentation or debugging purposes. The function either removes or improves on those items that are described in the following keyword arguments:

Arguments:

:unused-pathnames — If the value of **:unused-pathnames** is **t**, **sys:band-cleaner** garbage collects pathname objects that are no longer being used. This action is fairly lengthy because it must perform a **sys:full-gc**; however, it is necessary because the system accumulates numerous temporary pathnames that are not collected by incremental garbage collection. Both **sys:full-gc** and **sys:tree-shake** collect these temporary pathnames automatically; therefore, **:unused-pathnames** normally need not perform this action. Accordingly, the value of **:unused-pathnames** defaults to **nil**.

:pathname-properties — If the value of **:pathname-properties** is **t**, **sys:band-cleaner** deletes all property-list items from generic pathnames. Do not specify a value of **t** if you plan to use **make-system** or **delete-system** later.

If the value of **:pathname-properties** is **:make-system** (the default value), **sys:band-cleaner** deletes all property list items except for those used by **make-system**, **load-if**, and **delete-system** (to know which files have been loaded). The properties that are deleted is information such as the file attributes and compiler version.

If the value of **:pathname-properties** is **nil**, **sys:band-cleaner** preserves all pathname properties.

:cdr-code-plists — If the value of **:cdr-code-plists** is **t**, **sys:band-cleaner** ensures that all symbol property lists are **cdr**-coded. This saves memory space and speeds up access. The default value is **t**.

:previous-defs — If the value of **:previous-defs** is **t**, **sys:band-cleaner** removes any previous definitions of functions. The default value is **t**.

:debug-structures — If the value of **:debug-structures** is **t**, **sys:band-cleaner** entirely removes any function debug-info structures. The default value of this keyword is **nil**. Do not change the default value, because the debugger will not know the name of the function it is in, and may damage the environment beyond repair.

:doc-strings — If the value of **:doc-strings** is **t**, **sys:band-cleaner** removes documentation strings, descriptive arglists, function-parent, self-flavor, macros-expanded and value lists from the debug-info structures of functions and methods, and removes the documentation strings of variables and loaded patches. This option saves memory space, but it means that the functions **documentation** and **arglist** and the commands CTRL-SHIFT-A, CTRL-SHIFT-D, and CTRL-SHIFT-V will no longer be useful. Therefore, this option is usually desirable for an application delivery band, but not for a development band. The default value is **t**.

:zmacs-properties — If the value of **:zmacs-properties** is **t**, **sys:band-cleaner** removes source-file, zmacs-buffer, and zmacs-indentation properties meaning that the Zmacs command META-. will no longer work and indentation of Lisp source will not always be correct. Therefore, do this only for bands that do not include Zmacs. The default value depends on whether Zmacs is currently loaded.

:compiler-properties — If the value of **:compiler-properties** is **t**, **sys:band-cleaner** removes symbol properties and function debug-info properties that are only used by the compiler. After these items are removed, the compiler will no longer work; therefore, only use this option for a band that does not include the compiler. The default value depends on whether the compiler is currently loaded.

:arglists-and-local-maps — If the value of **:arglists-and-local-maps** is **t**, **sys:band-cleaner** removes arglist and local maps from the debug-info structures of functions, meaning that the debugger no longer will be able to display the names of arguments and local variables. The default value is **nil**.

:verbose — If the value of **:verbose** is **t**, **sys:band-cleaner** displays progress messages. The default value is **t**.

The sys:delete-system Function

A.4 The system deletion function is defined as follows:

sys:delete-system &optional *system* &key **:batch :keep-symbols :verbose**

The function **sys:delete-system** takes as its argument the name of a system or function to be undefined. A related function, **sys:unload-file**, undefines everything that was previously defined in a particular file or list of files.

Arguments:

system — Either a symbol or string which is the name of a system or feature (run the function without any arguments to see a list of meaningful system names), or a symbol which is the name of a function.

:batch — If the value of **:batch** is **t**, **sys:delete-system** suppresses all queries. The default value is **nil**.

:keep-symbols — This keyword can be followed by a list of symbols whose definitions are to be retained. Use this option to delete everything in a system *except* for particular pieces that you want to keep.

:verbose — If the value of **:verbose** is **t** (the default), **sys:delete-system** reports everything that it undefines.

For example, the following Lisp form would remove the Zmacs editor from a development band:

```
(sys:delete-system 'zmacs)
```

For an application band that does not need access to disk files, the loader and file system can be deleted with the following forms:

```
(sys:delete-system 'load) ; deletes loader
(sys:delete-system 'local-fs) ; local file system
```

Unless the **:batch** option is true, you are asked one or more questions to confirm what is to be done. Normally you answer all the questions YES. Files that are included in more than one system are not actually deleted until the last of the affected systems is deleted.

When using **sys:delete-system**, keep in mind the following:

A system name may be displayed as a meaningful argument; however, this does not mean that you can safely delete it. Exercise some judgment about what features you require. For example, if you use the Lisp Listener in a development band, and you delete the Input Editor system, you will discover that the Lisp Listener no longer works.

The following list illustrates some of the major dependencies between systems; each system shown requires the preceding systems that it is indented under. Some systems appear more than once because of multiple dependencies.

Window system
    Printer
        Imagen
    Streamer tape
    Profile
    UCL and Input Editor
        Telnet
            VT100
        Font-Editor
        Zmacs
            Debug Tools
            Mail Daemon
            Mail Reader
            Namespace Editor
            Visidoc
Pathname and Network support
    File-system
    Network service
    Namespace
        Namespace Editor
    IP or Chaosnet
        Telnet and VT100
        Mail Daemon
    NFS
Micronet
    RPC
        NFS

This is not intended to be a complete list of dependencies, and there are many more subtle interactions between systems as well.

The **sys:delete-system** tries its best to undo the effects of loading a system, but it is not perfect. It undefines everything that was defined using normal macros, but it cannot undo the effects of random top-level forms. As a result, some pieces may remain that will require manual cleanup.

Note that **sys:delete-system** only undefines items; memory space is not actually reclaimed until a full garbage collection is performed. Typically you would run **sys:delete-system** for each item to be deleted, and then run the **sys:tree-shake** function, which not only garbage-collects the function definitions, but also garbage-collects symbols that are no longer used. The variable **sys:*packages-to-be-cleaned*** is set by **sys:delete-system** to a list of packages from which definitions have been deleted. Therefore you can execute the following form to garbage collect the symbols and function definitions:

```
(sys:tree-shake :clean-packages sys:*packages-to-be-cleaned*)
```

Deleting UCL Commands

A.5 Applications that are based on UCL (such as the development band's Lisp Listener, Debugger, and Peek) can be customized by deleting unneeded commands. To do so, follow these steps:

1. Bring up the application window.
2. Press the HELP key.
3. Click on `Customization Menu` in the pop-up menu.
4. Click on `Command Editor`.
5. Point to the command to be deleted.
6. Click right to pop up a command menu, and then click on `Delete Command`.
7. Repeat the previous two steps as often as necessary.
8. When finished, press the END key.

NOTE: As with **sys:delete-system**, you must run **sys:tree-shake** before the memory space used by the command is actually reclaimed.

To delete commands from Zmacs, use **sys:delete-system** to delete the function that implements the command. To find the function name, press the HELP key while in Zmacs, and then press C followed by the command keystroke(s). This action displays a description in which the function name follows the words `implemented by`.

The sys:tree-shake Function

A.6 The name of the **sys:tree-shake** function is derived from the image of shaking a tree so that the dead leaves fall off. The function is basically for garbage-collecting symbols that are no longer being used (that is, which have no pointers to them), along with any values or definitions attached to those symbols. If you fail to run **sys:tree-shake**, a symbol cannot be deleted once it is interned.

sys:tree-shake &key :clean-packages :purge-packages :kill-packages :keep-packages :keep-symbols :undo-previous-training :batch :kamikaze

This function uninterns all symbols from the affected packages, performs a full garbage collection, and then reinterns the symbols that survived the garbage collection.

The following arguments are a package, a package name, or a list of packages:

Arguments:

:clean-packages — If you specify the **:clean-packages** keyword with a package, a package name, or a list of packages, **sys:tree-shake** deletes unused internal symbols in these packages.

:purge-packages — If you specify the **:purge-packages** keyword with a package, a package name, or a list of packages, **sys:tree-shake** deletes the unused symbols (both internal and external) in those packages.

:kill-packages — If you specify the **:kill-packages** keyword with a package, a package name, or a list of packages, **sys:tree-shake** deletes everything defined here even if they are used.

The **:kill-packages** option differs from using the **kill-package** function in that any symbols that are still referenced from outside the package remain interned in the package (although without any definitions attached) and the package itself is deleted only if it becomes empty. You should normally use **sys:delete-system** on the package first because it can perform additional application-specific clean up. On the other hand, **sys:tree-shake** can delete some items that **sys:delete-system** missed.

NOTE: If you do not specify any of the keywords above, then **sys:tree-shake** cleans all packages except for those specified by the **:keep-packages** argument.

:keep-packages — If you specify the **:keep-packages** keyword with a package, a package name, or a list of packages, **sys:tree-shake** will delete nothing from the specified packages.

:keep-symbols — The value of this keyword should be a list of symbols that are to be retained despite the actions requested by the previous arguments.

:undo-previous-training — The value of this keyword specifies whether to garbage-collect regions that have been made static by load-band training. If this is false, then any deleted functions that were part of the original product band will not be removed from memory. The default value is based on whether you request purging or killing any of the original system packages and whether **sys:delete-system** has undefined anything in those packages.

:batch — If the value of **:batch** is true, **sys:tree-shake** suppresses querying for confirmation. The default value causes **sys:tree-shake** to ask for confirmation before actually doing anything.

:kamikaze — If the value of **:kamikaze** is true, **sys:tree-shake** will delete itself before returning. The value of **:kamikaze** defaults to **nil**; however, you normally specify a value of **t**, except during experimentation when you might want to run the function more than once.

NOTE: Another full garbage collection is needed after running **sys:tree-shake** in order to reclaim the data structures it uses; this is normally performed as part of band training or by **sys:gc-and-disk-save**.

The following Lisp form allows you to clear dead wood from the internal implementation and still retain all documented features (the default action if **sys:tree-shake** is called without arguments):

```
(sys:tree-shake :clean-packages (list-all-packages))
```

If, on the other hand, the band is to be used only for running a particular application that is included in the band (and you will not be loading or evaluating new code), then the following Lisp form would remove everything that is not actually used by the application:

```
(sys:tree-shake :purge-packages (list-all-packages)
                :keep-symbols '(<my-function> ... ))
```

In this example, the `:keep-symbols` value is a list of the application program's entry point symbols, along with any other utilities you plan to run afterward. You probably want to keep **sys:start-training-session** and **sys:end-training-session** for later use. The function **sys:gc-and-disk-save** is retained automatically, and the read-eval-print loop, which invokes **sys:tree-shake**, is retained because it is currently in use.

Alternatively, the following Lisp form purges all packages other than the one containing the application:

```
(sys:tree-shake :purge-packages (list-all-packages)
                :keep-packages "<my-package>")
```

The sys:invert-tree Function

A.7 You can use the **sys:invert-tree** function in situations where an item does not get garbage collected and the cross-reference utility cannot find where it is referenced because the reference is in data rather than in code. This function scans the entire memory (similar to the garbage collector) to find where the pointers to a particular object are located. This action is sure to find all references, but running **sys:invert-tree** can be very time-consuming (from several minutes to a few hours, depending on the size of the load band and the depth of the reference chain).

NOTE: If you plan to use **sys:invert-tree**, call the **training-off** function *immediately* after login; otherwise, **sys:invert-tree** must first perform a full garbage collection to clear the training space regions prior to inverting the tree.

To load the **sys:invert-tree** function, execute the following form:

```
(load "sys:band-tools;invert-tree")
```

To use **sys:invert-tree**, call it with one argument, which is a list of objects to be found. Either the object itself or its address in memory can be included in the list.

For example, suppose that after performing all of the band shrinking operations, you find that one of the memory areas still seems larger than it should. You could use **describe-area** to get a list of region numbers and **sys:dump-objects-in-region** to see what is in each region. (If this function has already been deleted from the band, it can be loaded from file **sys:memory-management;memory-debug.xld.**) Suppose, for example, that the region display looks like this:

Address	Size	Object
1500000	10	"SYS::MAP-OBJECTS-IN-REGION"
...		
1500064	30	#<ART-Q-LIST-24 1500067>
1500114	30	#<ART-Q-LIST-24 1500117>

The following Lisp form can then find the pointers to these arrays:

```
(sys:invert-tree '(#o1500000 #o1500064 #o1500114))
```

The output written by **sys:invert-tree** to ***standard-output*** consists of one line for each link in the chain of references leading from some symbol (representing a global variable or function) to one of the specified objects. If the object being sought is a symbol, the pointer to it from its package will not be counted. Similarly, the pointers from symbols to their packages are not followed.

The following contents of a dribble file show an example of the output:

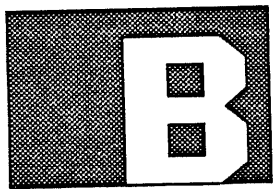
```
> (sys:invert-tree '(#o3567156))
  Target objects:
  #<ART-Q-3 3567156>

Pass 0 with max-gen 2 finished with 11. new ranges and 12. total ranges.
Pass 1 with max-gen 2 finished with 10. new ranges and 22. total ranges.
Pass 2 with max-gen 2 finished with 27. new ranges and 49. total ranges.
Pass 3 with max-gen 2 finished with 2. new ranges and 51. total ranges.
Pass 4 with max-gen 2 finished with 3. new ranges and 54. total ranges.
Pass 5 with max-gen 2 finished with 3. new ranges and 57. total ranges.
Pass 6 with max-gen 2 finished with 1. new ranges and 58. total ranges.
Pass 7 with max-gen 2 finished with 0. new ranges and 58. total ranges.
Pass 8 with max-gen 3 finished with 0. new ranges and 58. total ranges.

 1 THING value cell
  2 #<ART-Q-2 3567162> index 1.
   3 Cons at #o442244 : (A B ...)
    4 Cons at #o442245 : (B #<ART-Q-3 3567156> ...)
     5 Cons at #o442263 : (#<ART-Q-3 3567156> D ...)
       6 #<ART-Q-3 3567156>
```

This example asked to find pointers to the object at octal memory address 3567156; the addressed object is displayed under the heading `Target objects`, which indicates a three-element array. The next nine lines were written while the program scanned memory; the information is not particularly useful, but the messages provide reassurance that something really is actually happening during this time-consuming process. Finally, the answer is displayed. This reference chain states that symbol `THING`'s value cell points to an array, and element 1 of that array points to a chain of cons cells, leading to one whose **car** is the array being sought. In other words, this shows us that the array is part of a data structure which is the value of the special variable `THING`.

DOCUMENTER SYSTEM



Introduction

B.1 The Documenter system is a set of functions that are intended to make understanding and documenting Lisp programs easier. These functions collect information from the current Lisp environment in virtual memory and display it in textual form. There are functions for documenting an individual function or variable and for documenting everything defined in a package or system.

The following functions are user-callable; these are external symbols in the DOC package and are also installed in the USER package for convenience. Their names give a general idea of their purpose; all functions are described individually later in this document.

calls-who cross-reference show-call-tree inverse-tree show-forest
document document-function document-variable
document-type document-flavor document-keyword
document-package document-file document-system
build-xref-table dump-xref-table erase-xref-table
unused-in-package unused-in-file

This document neither describes the format of the displays produced by these functions nor shows examples, because the best way to see what they do is to try them out yourself. Their documentation strings should provide enough information to begin using them.

The documenter system is loaded by executing the following form:

```
(make-system 'documenter :noconfirm)
```

Call Trees

B.2 The function **calls-who** can be thought of as the inverse of **who-calls**; given a function or function name, it tells who that function references. This will include any functions, special variables, or keywords that you would see as operands in the disassembly of the function. It also lists the macros that were expanded. It does not list Lisp functions or sub-primitives that have been compiled into instructions. This function also works on interpreted functions, in which case it does not show functions that would normally be compiled into instructions. The argument may also be the pathname of a loaded XLD file; when used this way, it will report on what is used in the top-level forms of the file other than function and variable definitions.

show-call-tree is similar to **calls-who**, but besides showing the functions called, it shows what each of those functions calls, and so on, recursively. Note that it shows only function calls and not macros or variables, although it does show functions that are called by the expansion of macros. See the function's documentation string for an explanation of the special symbols used in the output. On a Lisp machine, such a call tree keeps going until it includes the whole world; so consequently, you usually need to set limits on how much you want to see.

Set these limits by using one or more of the three optional keyword arguments:

:package — Shows subtree only of functions in this package.

:filter — A predicate to be called on each function name to decide whether to show it. Defaults to showing functions of non-trivial size.

:depth — Number of levels of the tree to show (default 50). The depth is also limited by the width of the window.

The function **show-forest** displays calling trees in the same format as **show-call-tree**, but instead of giving it a function to use as the root, its argument is either a system name or a list of filenames. **show-forest** will display calling trees for all of the functions defined in the designated files, automatically selecting which ones should be used as roots. This function is very helpful for quickly seeing how the functions of a program fit together.

The function **inverse-tree** takes a symbol to be used as the root and shows the tree of things that reference it. To keep the tree small enough to be useful, external symbols are considered leaves, as are symbols used in many places, but all known direct references to the root are included.

Document Functions

B.3 There are a series of functions for writing documentation about various items. For example, **document-function** takes a function or function name as its argument and tells you a variety of information about that function, including what file it is defined in, the parameter list, what functions it calls, what special variables it uses, who calls it, its documentation string, and so on. This could be thought of as a superset of the information shown by **calls-who** except that while **calls-who** defaults to showing everything that is referenced, the **document-function** default leaves out common trivial functions. For a macro or inline function, **document-function** will tell which functions contain an expansion of it. See the following paragraph for a further discussion of the called-by information. Similarly, there are the following functions:

- **document-variable** — For describing special variables and instance variables.
- **document-type** — For describing data types defined by **deftype** or **defstruct**.
- **document-flavor** — For describing flavors and optionally describing all of their methods and instance variables.
- **document-keyword** — For reporting where a keyword or other symbol is used as a constant.
- **document-package** — For invoking the preceding functions to describe all of the types, special variables, functions and macros, and flavors defined in a package. The names are listed alphabetically within each category.
- **document-file** — For describing everything defined in a file or list of files. Note that since all of these functions collect information from the object code in memory, the file must already be loaded.
- **document-system** — For describing everything defined in the files in a **defsystem**.

Most general of all is the **document** function, which will try to describe any object it is given. This is mainly useful for listing all of the definitions attached to a symbol. `(document #'foo)` is equivalent to `(document-function 'foo)`. `(document (find-package foo))` does the same thing as `(document-package foo)`. Given a string as an argument, **document** will call either **document-system**, **document-package**, or **document-file**, depending on whichever seems most meaningful.

Since the output from **document**, **document-package**, **document-file**, and **document-system** can all be rather lengthy, they all accept an output file pathname as an optional second argument; if supplied, the output is written there instead of to ***standard-output***. **document-package** binds ***package*** so that the symbols will be written without the package prefix. For **document-file** or **document-system**, you will probably want to set or bind ***package*** yourself before running the function.

Cross-Reference Table

B.4 Some discussion needs to be given to how the functions **document-function**, **document-variable**, and so on, discover who uses the object. This information is obtained from a cross-reference table which should normally be built before calling one of these functions, unless you are not interested in who uses it. Building the table takes some time, but once it is built, subsequent use of it is quite fast. The function **build-xref-table** is used to add data to the cross-reference table; the arguments are used to specify where it should look, and it will record in the table all of the references it finds in the functions that it looks at. If **build-xref-table** is called with no arguments, it scans all packages that have not already been scanned; this is not recommended since it takes about 30 minutes on a microExplorer running the Development System Software. Usually you would use the **:package** keyword argument to specify the package name or list of package names that you want scanned.

Other optional keyword arguments are:

:file — A pathname or list of pathnames; scan functions defined there. The file must have already been loaded. The `".lisp"` version of the file will be read to find the names of the functions defined there, but the definitions currently in memory are the ones that will actually be examined.

:directory — Scan functions defined in files loaded from this directory.

:system — Scan all files in this system or list of systems.

build-xref-table may be called as many times as you wish. For example, the single call `(build-xref-table :package 'foo :file "bar")` will have the same effect as the two calls `(build-xref-table :package 'foo)` and `(build-xref-table :file "bar")`.

The **document-package** function automatically builds the cross-reference table for the package being documented if it has not already been done, but you may want to use **build-xref-table** first to collect references from other places as well. Similarly, **document-file** and **document-system** will scan the files being documented for references if that has not already been done. **document-function** and **document-variable** will ask you if you want to build the cross-reference table for the symbol's package if not already done; this can be suppressed by using the optional keyword argument **:no-query**.

The function **erase-xref-table** can be called to clear the cross-reference table in case you either want to start over from scratch, or are finished with the table and want it to be garbage-collected. **erase-xref-table** is called automatically on a **full-gc**; if you want the table to be saved across a **full-gc**, then do:

```
(delete-initialization "Discard cross-reference table"
                       nil
                      'sys:full-gc-initialization-list)
```

The function **dump-xref-table** can be called to write the cross-reference table to an XLD file, which can be loaded later to restore the data (just use the **load** function). Loading the data from a file takes about half as long as building it initially. Note, however, that while **build-xref-table** adds data to the table, loading a file written by **dump-xref-table** replaces whatever data was there before. You can, however, load a file, call **build-xref-table** to add more data, and then call **dump-xref-table** to write a new file.

doc:cross-reference Function

B.5 For cases where the output produced by **document-package**, **document-file**, or **document-system** is more verbose than desired, a briefer cross-reference listing is provided by the function **doc:cross-reference**. For each symbol that names a function, variable, or type, it shows the name of the file in which it is defined, and the names of functions or methods that reference it. The following keyword arguments specify which definitions are to be included; each can be either a single item or a list.

:package — A package or package name — include local symbols of the package.

:file — A pathname or filename string — include things defined there.

:system — A symbol or string naming a system — include things defined there.

If more than one of these is specified, the union of the sets is used. If none are specified, then whatever definitions are already in the cross-reference table are used. Two more optional keywords control the output:

:output-file — Write the report to this file or to ***standard-output***.

:line-length — Maximum number of characters per line of output. When writing to a window, this defaults to the width of the window.

Finding Unused Definitions

B.6 The function **unused-in-package** can be used to generate a report of the things that are defined in a package but are not used. Similarly, **unused-in-file** will identify things that are defined in a file but are not used, either directly or indirectly, by anything outside of the file.